Unitex - Manuel d'utilisation

Sébastien Paumier

UNITEX 1.2

USER MANUAL

Université de Marne-la-Vallée

http://www-igm.univ-mlv.fr/~unitex
unitex@univ-mlv.fr

Sébastien Paumier - June 2006

English translation by the local grammar group at the
CIS, Ludwig-Maximilians-Universität, Munich - Oct 2003
(Wolfgang Flury, Franz Guenthner, Friederike Malchok, Clemens Marschner, Sebastian Nagel, Johannes Stiehler)
http://www.cis.uni-muenchen.de/

Contents

Introduction

Unitex is a collection of programs developped for the analysis of texts in natural languages by using linguistic resources and tools. These resources consist of electronic dictionaries, grammars and lexical grammar tables, initially developed for French by Maurice Gross and his students at the Laboratoire d'Automatique Documentaire et Linguistique (LADL) . Similar resources have been developed for other languages in the context of the RELEX laboratory network.

The electronic dictionaries specify the simple and compound words of a language together with their lemmas and a set of grammatical (semantic and inflectional) codes. The availability of these dictionaries is a major advantage compared to the usual utilities for pattern searching as the information they contain can be used for searching and matching, thus describing large classes of words using very simple patterns. The dictionaries are presented in the DELA formalism and were constructed by teams of linguists for several languages (French, English, Greek, Italian, Spanish, German, Thai, Korean, Polish, Norwegian, Portuguese, etc.)

The grammars deployed here are representations of linguistic phenomena on the basis of recursive transition networks (RTN), a formalism closely related to finite state automata. Numerous studies have shown the adequacy of automata for linguistic problems at all descriptive levels from morphology and syntax to phonetic issues. Grammars created with Unitex carry this approach further by using a formalism even more powerful than automata. These grammars are represented as graphs that the user can easily create and update.

Tables built in the context of lexicon-grammar are matrices describing properties of certain words. Many such tables have been constructed for all simple verbs in French as a way of describing their relevant properties. Experience has shown that every word has a quasi-unique behavior, and these tables are a way of presenting the grammar of every element in the lexicon, hence the name lexicon-grammar for this linguistic theory. Unitex offers a way to directly construct grammars from these tables.

Unitex can be viewed as tool with which one can put these linguistic resources to use. Its technical characteristics are its portability, modularity, the possibility of dealing with languages that use special writing systems (e.g. many Asian languages), and its openness, thanks to its open source distribution. Its linguistic characteristics are the ones that have motivated the elaboration of these resources: precision, completeness, and the taking into account of frozen expressions, most notably those which concern the enumeration of compound words.

The first chapter describes how to install and run Unitex.

Chapter 2 presents the different steps in the analysis of a text.

Chapter 3 describes the formalism of the DELA electronic dictionaries and the different operations that can be applied to them.

Chapters 4 and 5 present different means for making text searches more effective. Chapter 5 describes in detail how the graph editor is used.

Chapter 6 is concerned with the different possible applications of grammars. The particularities of each type of grammar are presented.

Chapter 7 introduces the concept of a text automaton and describes the properties of this notion. This chapter also describes the operations on this object, in particular, how to disambiguate lexical items with the ELAG program.

Chapter 8 contains an introduction to lexicon-grammar tables, followed by a description of the method of constructing grammars based on these tables.

Chapter 9 contains a detailed description of the different external programs that make up the Unitex system.

Chapter 10 contains descriptions of all file formats used in the system.

The reader will find in appendix the GPL and LGPL licenses under which the Unitex source code is released, as well as the LGPLLR license which applies for the linguistic data distributed with Unitex.

Chapter 1

Installation of Unitex

Unitex is a multi-platform system that runs on Windows as well as on Linux or MacOS. This chapter describes how to install and how to launch Unitex on any of these systems. It also presents the procedures used to add new languages and to uninstall Unitex.

1.1 Licenses

Unitex is a free software. This means that the sources of the programs are distributed with the software, and that anyone can modify and redistribute them. The code of the Unitex programs is under the LGPL licence ([24]), except for the TRE library for dealing with regular expressions from Ville Laurikari ([36]), which is under GPL licence ([23]). The LGPL Licence is more permissive than the GPL licence, because it makes it possible to use LGPL code in nonfree software. From the point of view of the user, there is no difference, because in both cases, the software can freely be used and distributed.

All the data that go with Unitex are distributed under the LGPLLR license ([29]).

Full text versions of GPL, LGPL and LGPLLR can be found in the appendices of this manual.

1.2 Java runtime environment

Unitex consists of a graphical interface written in Java and external programs written in *C/C++*. This mixture of programming languages is responsible for a fast and portable application that runs on different operating systems.

Before you can use the graphical interface, you first have to install the runtime environment, usually called Java virtual machine or JRE (Java Runtime Environment).

For the graphical mode, Unitex needs Java version 1.4 (or newer). If you have an older version of Java, Unitex will stop after you have chosen the working language.

You can download the virtual machine for your operating system for free from the Sun Microsystems web site ([38]) at the following address: `http://java.sun.com`.

If you are working under Linux or MacOS, or if you are using a Windows version with personal user accounts, you have to ask your system administrator to install Java.

1.3 Installation on Windows

If Unitex is to be installed on a multi-user Windows machine, it is recommended that the systems administrator performs the installation. If you are the only user on your machine, you can perform the installation yourself.

Decompress the file `unitex_1.2.zip` (You can download this file from the following address: http://www-igm.univ-mlv.fr/~unitex) into a directory `Unitex` that should preferably be created within the `Program Files` folder.

After decompressing the file, the `Unitex` directory contains several subdirectories one of which is called `App`. This directory contains a file called `Unitex.jar`. This file is the Java executable that launches the graphical interface. You can double click on this icon to start the program. To facilitate launching Unitex, you may want to add a shortcut to this file on the desktop.

1.4 Installation on Linux and Mac OS X

In order to install Unitex on Linux, it is recommended to have system administrator permissions. Decompress the file `unitex_1.2.zip` in a directory named `Unitex`, by using the following command:

```
unzip Unitex_1.2.zip -d Unitex
```

Within the directory `Unitex/Src/C++`, start the compilation of Unitex with the command:

```
make install
```

You can then create an alias in the following way:

```
alias unitex='cd /..../Unitex/App/ ; java -jar Unitex.jar'
```

1.5 First use

If you are working on Windows, the program will ask you to choose a personal working directory, which you can change later in "Info>Preferences...>Directories". To create a directory, click on the icon showing a file (see figure 1.3).

If your are using Linux or MacOS, the program will automatically create a `/unitex` directory in your `$HOME` directory. This directory allows you to save your personal data. For each language that you will be using, the program will copy a root directory of that language to your peronal work directory, except the dictionaries. You can also modify your copy of the files without risking to damage the system files.

Figure 1.1: First use under Windows

Figure 1.2: First use under Linux

1.6 Adding new languages

There are two different ways to add languages. If you want to add a language that is to be accessible by all users, you have to copy the corresponding directory to the `Unitex` system directory, for which you will need to have the access rights (this might mean that you need to ask your system administrator to do it). On the other hand, if the language is only used by a single user, he can also copy the directory to his working directory. He can work with this language without this language being shown to other users.

1.7 Uninstalling Unitex

No matter which operating system you are working with, it is sufficient to delete the `Unitex` directory to completely delete all the program files. Under Windows you may have to delete

Figure 1.3: Creating the personal work directory

the shortcut to `Unitex.jar` if you have created one on your desktop. The same has to be done on Linux, if you have created an alias.

Chapter 2

Loading a text

One of the main functionalities of Unitex is to search a text for expressions. For this, texts have to undergo a set of preprocessing steps that normalize non-ambiguous forms and split the text in sentences. Once these operations are performed, the electronic dictionaries are applied to the texts. Then one can search more effectively in the texts by using grammars.

This chapter describes the different steps for text preprocessing.

2.1 Selecting a language

When starting Unitex, the program asks you to choose the language in which you want to work (see figure 2.1). The languages displayed are the ones that are present in the `Unitex` system directory and those that are installed in your personal working directory. If you use a language for the first time, Unitex copies the system directory for this language to your personal directory, except for the dictionaries in order to save disk space. WARNING: if you already have a personal directory for a given language, Unitex won't try to copy system data into it. So, if an update has modified a resource file other than a dictionary, you will have to copy by yourself this file, or to delete your personal directory for this language, and let Unitex rebuild it properly.

Choosing the language allows Unitex to find certain files, for example the alphabet file. You can change the language at any time by choosing "Change Language..." in the "Text" menu. If you change the language, the program will close all windows related to the current text, if there are any. The active language is indicated in the title bar of the graphical interface.

2.2 Text formats

Unitex works with Unicode texts. Unicode is a standard that describes a universal character code. Each character is given a unique number, which allows for representing texts without having to take into account the proprietary codes on different machines and/or operating systems. Unitex uses a two-byte representation of the Unicode 3.0 standard, called Unicode Little-Endian (for more details, see [12]).

Figure 2.1: Language selection when starting Unitex

The texts that come with Unitex are already in Unicode format. If you try to open a text that is not in the Unicode format, the program proposes to convert it automatically (see figure 2.2).

This conversion is based on the current language: if you are working in French, Unitex proposes to convert your text [1] assuming that it is coded using a French code page. By default, Unitex proposes to either replace the original text or to rename the original file by inserting `.old` at the beginning of its extension. For example, if one has an ASCII file named `balzac.txt`, the conversion process will create a copy of this ASCII file named `balzac.old.txt`, and will replace the contents of `balzac.txt` with its equivalent in Unicode.

Figure 2.2: Automatic conversion of a non-Unicode text

If the encoding suggested by default is not correct or if you want to rename the file differently than with the suffix `.old`, you can use the "Transcode Files" command in the "File Edition" menu. This command enables you to choose source and target encodings of

[1] Unitex also proposes to automatically convert graphs and dictionaries that are not in Unicode Little-Endian.

the documents to be converted (see figure 2.3). By default, the proposed source encoding is that which corresponds to the current language and the destination encoding is Unicode Little-Endian. You can modify these choices by selecting any source and target encoding. Thus, if you wish, you can convert your data into other encodings, as for example UTF-8 in order for instance to create web pages. The button "Add Files" enables you to select the files to be converted. The button "Remove Files" makes it possible to remove a list of files erroneously selected. The button "Transcode" will start the conversion of all the selected files. If an error occurs when the file is processed (for example, a file which is already in Unicode), the conversion continues with the next file.

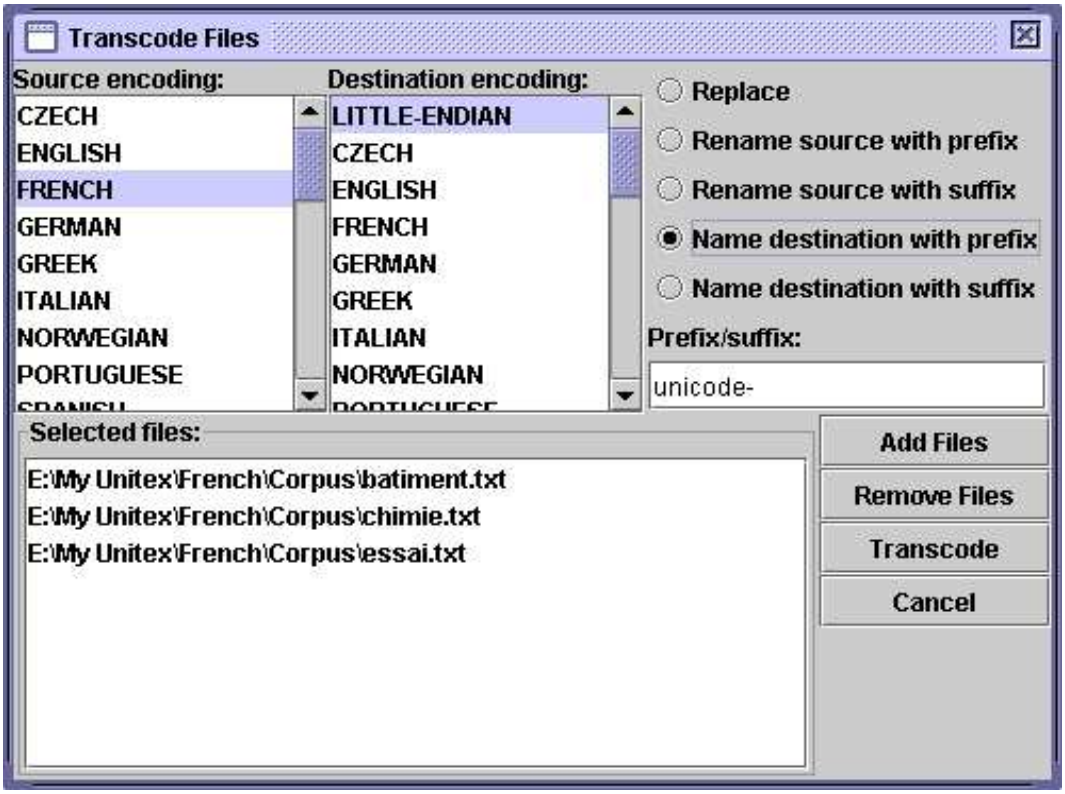


Figure 2.3: File conversion

To obtain a text in the right format, you can also use a text processor like the free software from OpenOffice.org ([41]) or Microsoft Word, and save your document with the format "Unicode text".

In Office XP, you have to choose the "Raw Text (*.txt)" format and then select the "Unicode" encoding in the configuration window as shown on figure 2.4.

By default, the encoding proposed on a PC is always Unicode Little-Endian. The texts thus obtained do not contain any formatting information anymore (fonts, colors , etc.) and are ready to be used with Unitex.

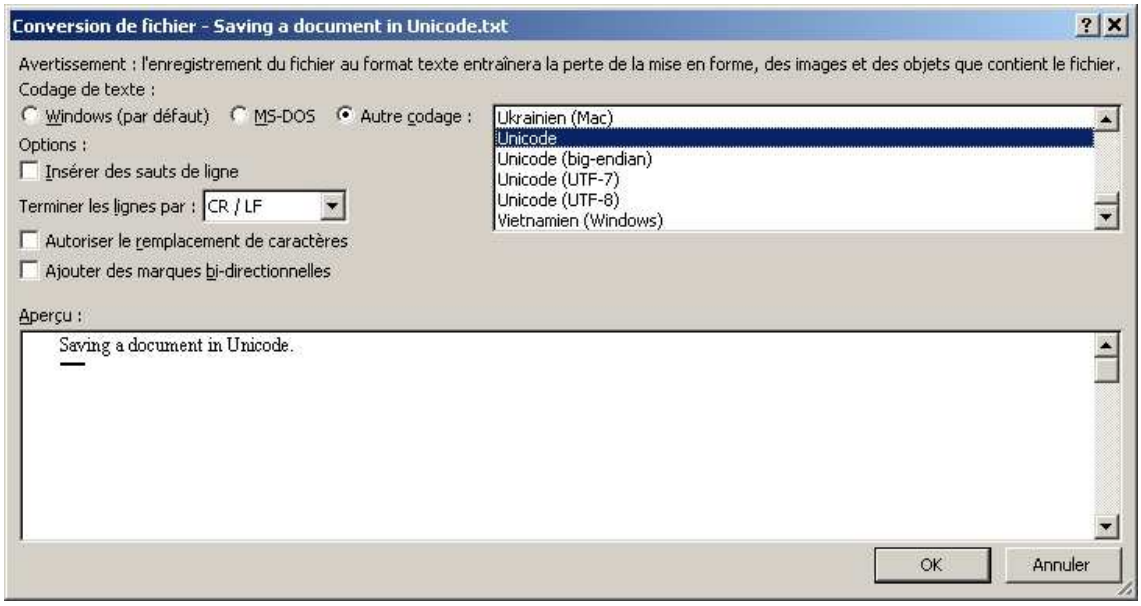


Figure 2.4: Saving in Unicode with Office XP

2.3 Editing text files

You also have the possibility of using the text editor integrated into Unitex, accessible via the "Open..." command in the "File Edition" menu". This editor offers search and replace functionalities for the texts and dictionaries handled by Unitex. To use it, click on the "Find" icon. You will then see a window divided into three parts. The "Find" part corresponds to the usual search operations. If you open a text split into sentences, you will have the possibility to search by the number of a sentence in the "Find Sentence" part. Lastly, the "Search Dictionary" part, visible in figure 2.5, enables you to carry out operations concerning the electronic dictionaries. In particular, you can search by specifying if it concerns inflected terms, lemmas, the grammatical and semantic and/or the inflecional codes. Thus, if you want to search for all the verbs which have the semantic feature `t`, which indicates transitivity, it is enough to search for `t` by clicking on "Grammatical code". You will get the matching entries without confusion with all the other occurrences of the letter `t`.

2.4 Opening a text

Unitex deals with two types of text files.

The files with the extension `.snt` are text files preprocessed by Unitex which are ready to be manipulated by the different system functions. The files ending with `.txt` are raw files.

To use a text, open the `.txt` file by clicking on "Open..." in the "Text" menu.

Choose the file type "Raw Unicode Texts" and select your text. Files larger than 2 MBytes are not displayed. The message "`This file is too large to be displayed. Use`

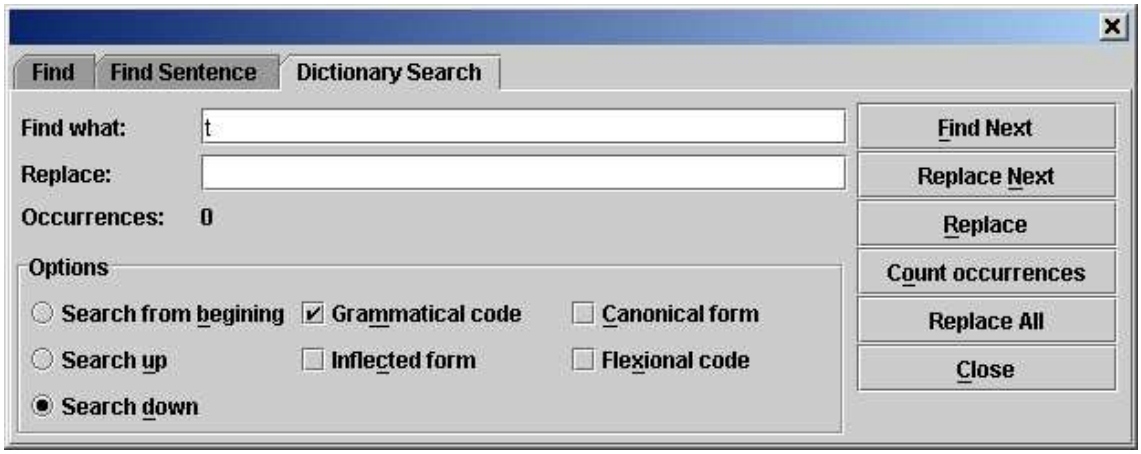


Figure 2.5: Searching an electronic dictionary for the semantic feature `t`

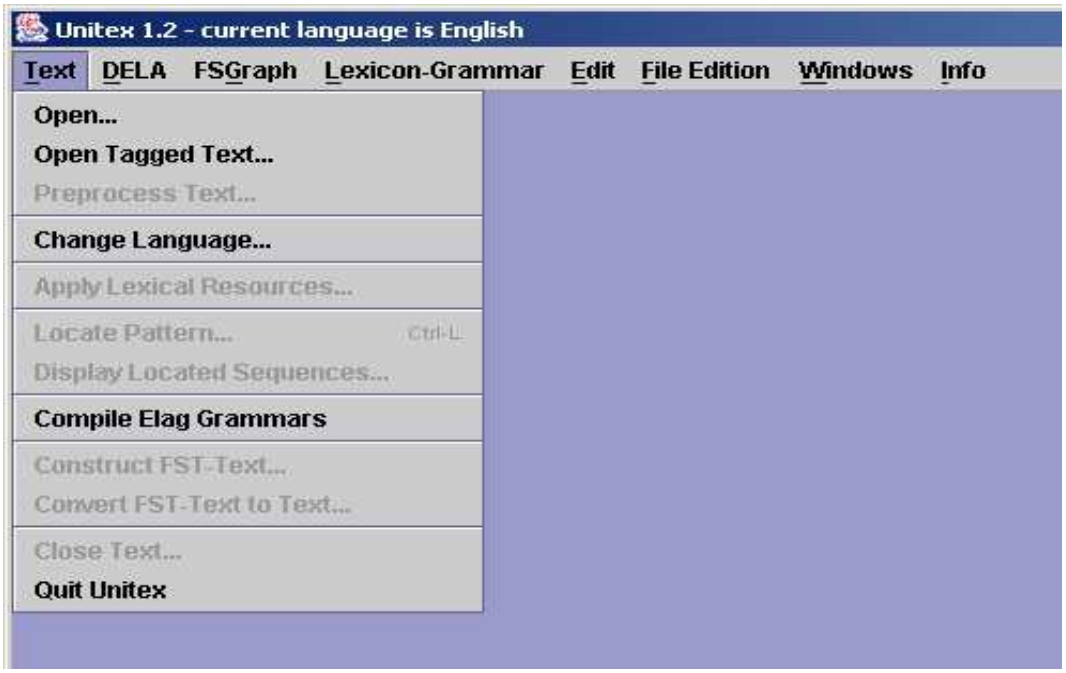


Figure 2.6: Text Menu

`a word processor to view it.`" is displayed in the window. This limit applies to all open text files (the list of tokens, dictionaries, etc.). To modify this limit, use the menu "Info>Preferences" and set the new value for "Maximum Text File Size" in the tab "Text Presentation" (see 4.7, page 62).

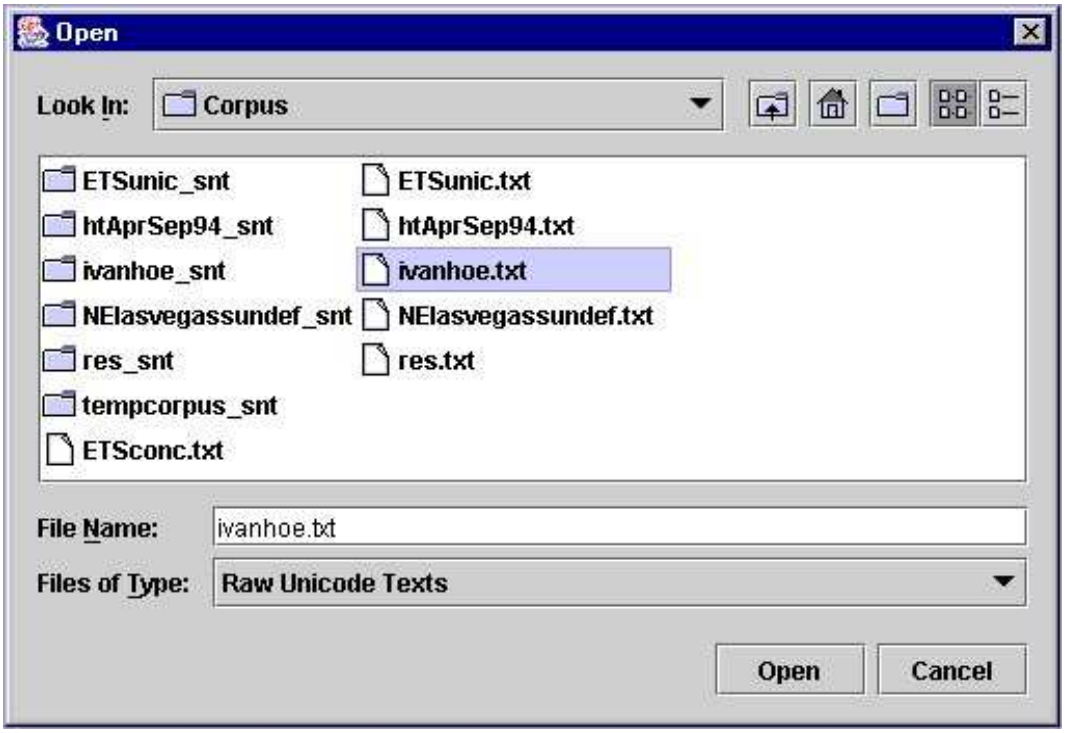


Figure 2.7: Opening a Unicode text

2.5 Preprocessing a text

After a text is selected, Unitex offers to preprocess it. Text preprocessing consists of performing the following operations: Normalization of separators, identification of tokens, normalization of non-ambiguous forms, splitting into sentences and the application of dictionaries. If you choose not to preprocess the text, it will nevertheless be normalized and tokens will be looked up, since these operations are necessary for all further Unitex operations. It is always possible to carry out the preprocessing later by clicking on "Preprocess Text..." in the "Text" menu.

If you choose to preprocess the text, Unitex proposes to parameterize it as in the window shown in figure 2.8.

The option "Apply FST2 in MERGE mode" is used to split the text into sentences. The option "Apply FST2 in REPLACE mode" is used to make replacements in the text, especially for the normalization of non-ambiguous forms. With the option "Apply All default Dictionaries" you can apply dictionaries in the DELA format (Dictionnaires Electroniques du LADL). The option "Analyze unknown words as free compound words" is used in Norwegian for correctly analyzing compound words constructed via concatenation of simple forms. Finally, the option "Construct Text Automaton" is used to build the text automaton. This option is deactivated by default, because it consumes a large amount of memory and disk space if the text is too large. The construction of the text automaton is described in

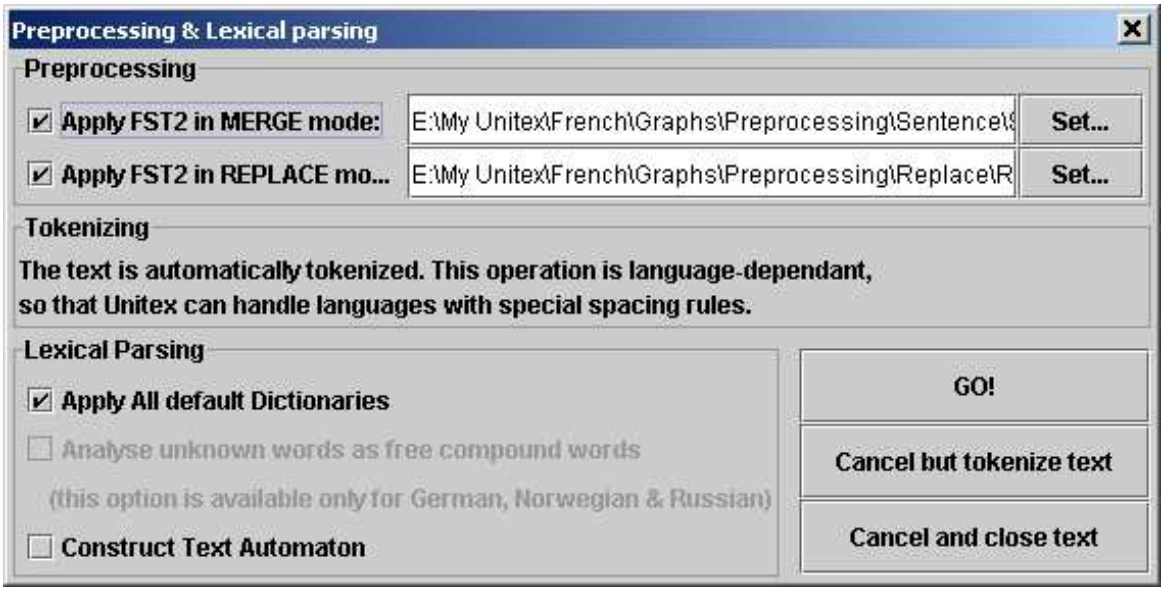


Figure 2.8: Preprocessing Window

chapter 7.

NOTE: If you click on "Cancel but tokenize text", the program will carry out the normalization of separators and split the text into tokens. Click on "Cancel and close text" to cancel the operation.

2.5.1 Normalization of separators

The standard separators are the space, the tab and the newline characters. There can be several separators following each other, but since this isn't useful for linguistic analyses, separators are normalized according to the following rules:

- a sequence of separators that contains at least one newline is replaced by a single newline
- all other sequences of separators are replaced by a space.

The distinction between space and newline is maintained at this point because the presence of newlines may have an effect on the process of splitting the text into sentences. The result of the normalization of a text named `my_text.txt` is a file in the same directory as the `.txt` file and is named `my_text.snt`.

NOTE: When the text is preprocessed using the graphical interface, a directory named `my_text_snt` is created immediately after normalization. This directory, called text directory, contains all the data associated with this text.

2.5.2 Splitting into sentences

Splitting texts into sentences is an important preprocessing step since this helps in determining the units for linguistic processing. The splitting is used by the text automaton construction program. In contrast to what one might think, detecting sentence boundaries is not a trivial problem. Consider the following text:

The family has urgently called Dr. Martin.

The full stop that follows *Dr* is followed by a word beginning with a capital letter. Thus it may be considered as the end of the sentence, which would be wrong. To avoid the kind of problems caused by the ambiguous use of punctuation, grammars are used to describe the different contexts for the end of a sentence. Figure 2.9 shows an example grammar for sentence splitting (for French sentences).

When a path of the grammar recognizes a sequence in the text and when this path produces the sentence delimiter symbol `{S}`, this symbol is inserted into the text.

The path shown at the top of figure 2.9 recognizes the sequence consisting of a question mark and a word beginning with a capital letter and inserts the symbol `{S}` between the question mark and the following word. The following text:

What time is it? Eight o' clock.

will be converted to:

What time is it ?{S} Eight o' clock.

A grammar for end-of-sentence detection may use the following special symbols:

- <E> : empty word, or epsilon. Recognizes the empty sequence;
- <MOT> : recognizes any sequence of letters;
- <MIN> : recognizes any sequence of letters in lower case;
- <MAJ> : recognizes any sequence of letters in upper case;
- <PRE> : recognizes any sequence of letters that begins with an upper case letter;
- <NB> : recognizes any sequence of digits (1234 is recognized but not 1 234);
- <PNC> : recognizes the punctuation symbols `; , ! ? :` and the inverted exclamation points and question marks in Spanish and some Asian punctuation letters;
- <^> : recognizes a newline;
- `#` : prohibits the presence of a space.

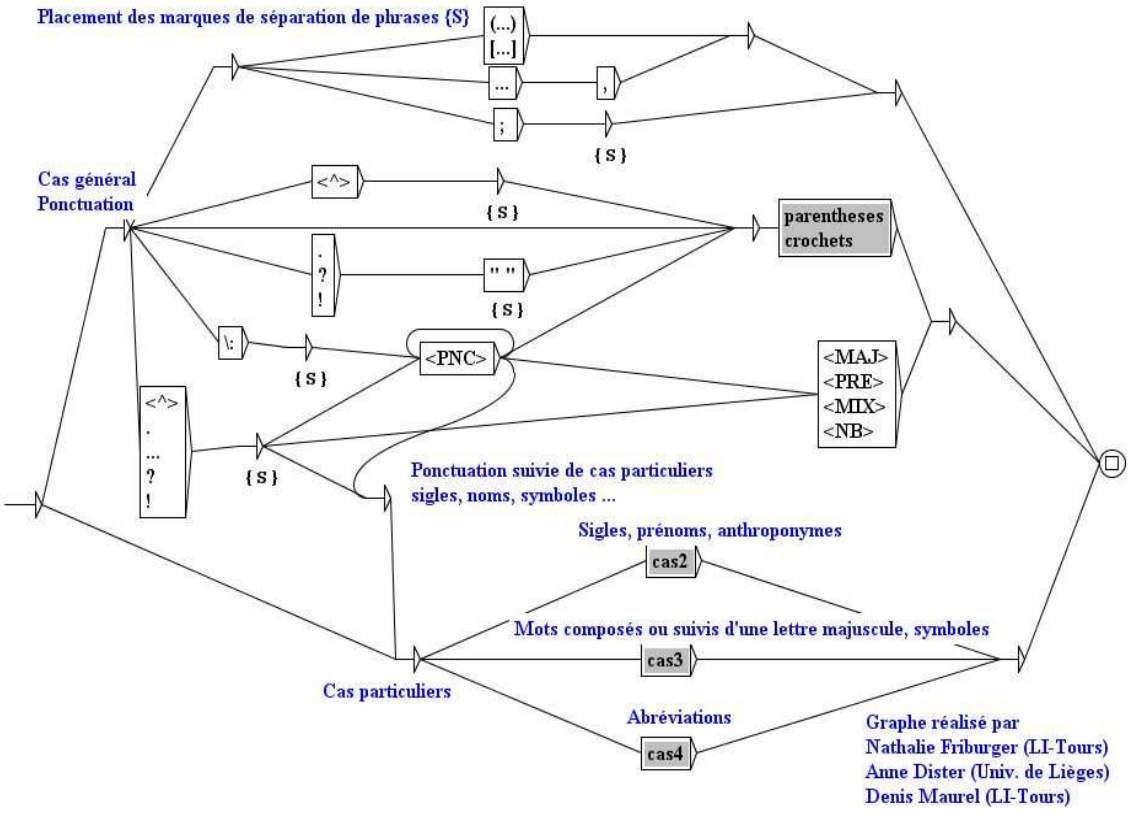


Figure 2.9: Sentence splitting grammar for French

By default, the space is optional between two boxes. If you want to prohibit the presence of the space you have to use the special character `#`. In contrary, if you want to force the presence of the space, you must use the sequence `" "`. Lower and upper case letters are defined by an alphabet file (see chapter 10). For more details on grammars, see chapter 5. For more information about sentence boundary detection, see [16]. The grammar used here is named `Sentence.fst2` and can be found in the following directory:

```
/(user home directory)/(language)/Graphs/Preprocessing/Sentence
```

This grammar is applied to a text with the `Fst2Txt` program in MERGE mode. This has the effect that the output produced by the grammar, in this case the symbol `{S}`, is inserted into the text. This program takes a `.snt` file and modifies it.

2.5.3 Normalization of non-ambiguous forms

Certain forms present in texts can be normalized (for example, the English sequence "*I'm*" is equivalent to "*I am*"). You may want to replace these forms according to your own needs. However, you have to be careful that the forms normalized are unambiguous or that the removal of ambiguity has no undesirable consequences.

For instance, if you want to normalize "*O'clock*" to "*on the clock*", it would be a bad idea to replace "*O'*" by "*on the* ", because a sentence like:

John O'Connor said: "it's 8 O'clock"

would be replaced by the following incorrect sentence:

John on the Connor said: "it's 8 on the clock"

Thus, one needs to be very careful when using the normalization grammar.

One needs to pay attention to spaces as well. For example, if one replaces "'re" by "are", the sentence:

You're stronger than him.

will be replaced by:

Youare stronger than him.

To avoid this problem, one should explicitly insert a space, i.e. replace "*'re*" by "*are*".

The accepted symbols for the normalization grammar are the same as the ones allowed for the sentence splitting grammar. The normalization grammar is called `Replace.fst2` and can be found in the following directory:

```
/(home directory)/(active language)/Graphs/Preprocessing/Replace
```

As in the case of sentence splitting, this grammar is applied using the `Fst2Txt` program, but in REPLACE mode, which means that input sequences recognized by the grammar are replaced by the output sequences that are produced. Figure 2.10 shows a grammar that normalizes verbal contractions in English.

2.5.4 Splitting a text into words (tokens)

Some languages, in particular Asian languages, use separators that are different from the ones used in western languages. Spaces can be forbidden, optional, or mandatory. In order to better cope with these particularities, Unitex splits texts in a language dependent way.

Thus, languages like French are treated as follows:

A token can be:

- the sentence delimiter `{S}`;
- the stop marker `{STOP}`. This token is a special one that can NEVER be matched in any way by a grammar. It can be used to bound elements in a corpus. For instance, if a corpus is made of news separated by `{STOP}`, it will be impossible that a grammar matches a sequence that overlaps the end of a news and the beginning of the following news;

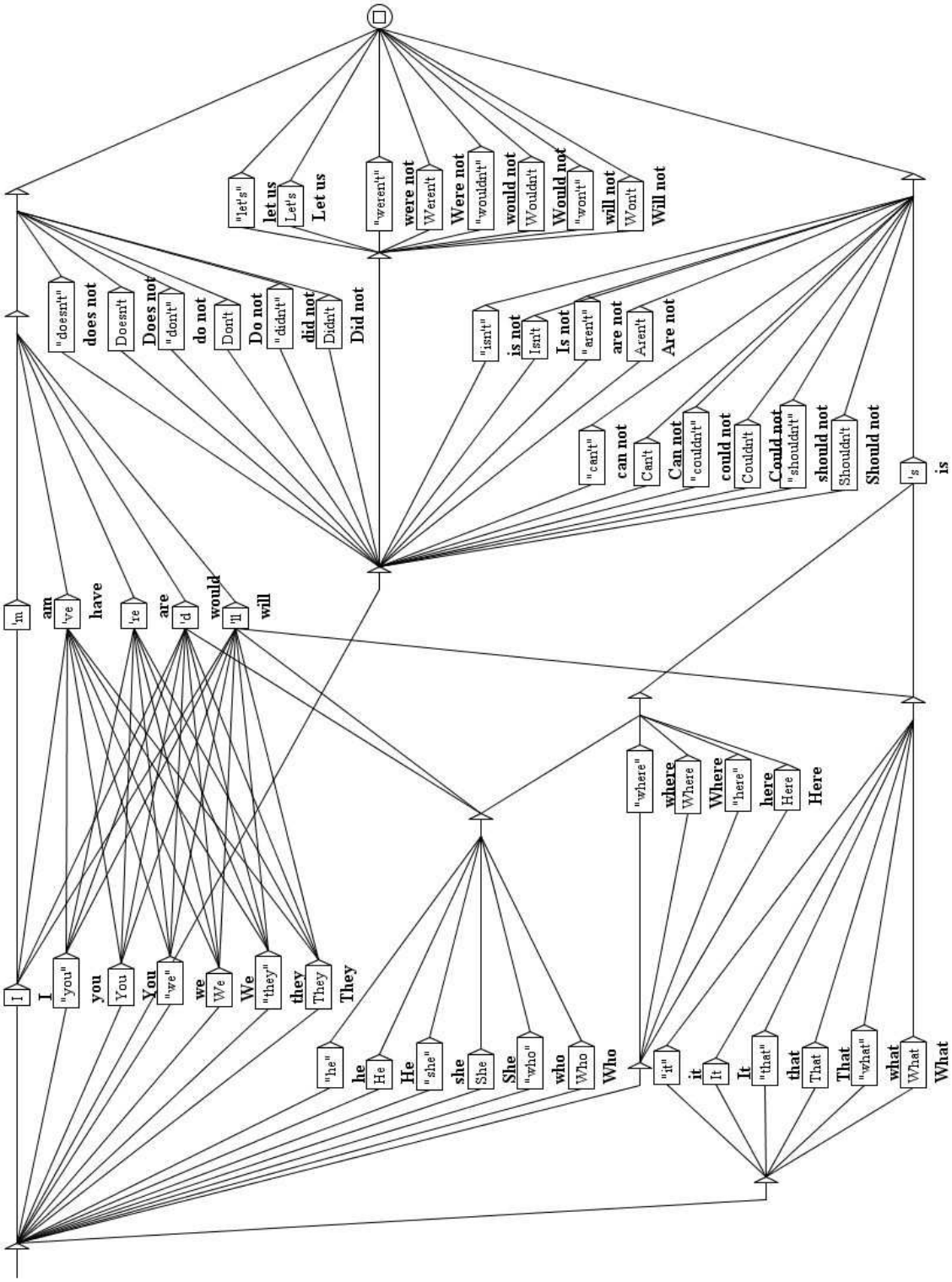


Figure 2.10: Normalization of English verbal contractions

- a lexical tag `{aujourd'hui,.ADV}`;
- a contiguous sequence of letters (the letters are defined in the language alphabet file);
- one (and only one) non-word character, i.e. all characters not defined in the alphabet file of the current language; if it is a newline, it is replaced by a space.

For other languages, tokenization is done on a character by character basis, except for the sentence delimiter `{S}`, the `{STOP}` marker and lexical tags. This simple tokenization is fundamental for the use of Unitex, but limits the optimization of search operations for patterns.

Regardless of the mechanism used, the newlines in a text are replaced by spaces. Tokenization is done by the `Tokenize` program. This program creates several files that are saved in the text directory:

- `tokens.txt` contains the list of tokens in the order in which they are found in the text;
- `text.cod` contains an integer array; every integer corresponds to the index of a token in the file `tokens.txt`;
- `tok_by_freq.txt` contains the list of tokens sorted by frequency;
- `tok_by_alph.txt` contains the list of tokens in alphabetical order;
- `stats.n` contains some statistics about the text.

Tokenizing the text:

A cat is a cat.

returns the following list of tokens: *A* SPACE *cat is a .*

You will observe that tokenization is case sensitive (A and a are two distinct tokens), and that each token is listed only once. Numbering these tokens with 0 to 5, the text can be represented by a sequence of numbers (integers) as described in the following table:

Token number	0	1	2	1	3	1	4	1	2	5
Corresponding token	*A*		*cat*		*is*		*a*		*cat*	.

Table 2.1: Representation of the text *A cat is a cat.*

For more details, see chapter 10.

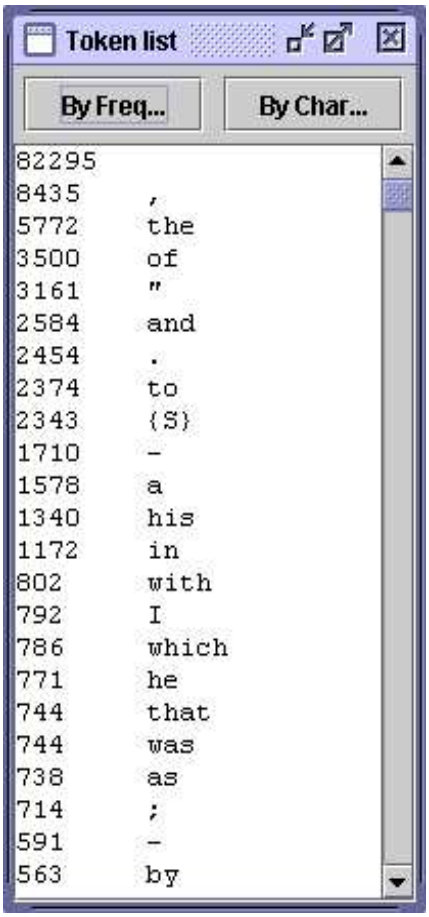


Figure 2.11: Tokens of an English text sorted by frequency

2.5.5 Applying dictionaries

Applying dictionaries consists of building the subset of dictionaries consisting only of forms that are present in the text. Thus, the result of applying a English dictionary to the text *Igor's father in law is ill* produces a dictionary of the following simple words:

```
father,.N+Hum:s
father,.V:W:P1s:P2s:P1p:P2p:P3p
ill,.A
ill,.ADV
ill,.N:s
in,.A
in,.N:s
in,.PART
in,.PREP
is,be.V:P3s
```

```
is,i.N:p
law,.N:s
law,.V:W:P1s:P2s:P1p:P2p:P3p
s,.N:s
```

as well as a dictionary of compound words consisting of a single entry:

```
father in law,.N+NPN+Hum+z1:s
```

Since the sequence *Igor* is neither a simple English word nor a part of a compound word, it is treated as an unknown word. The application of dictionaries is done through the program `Dico`. The three files produced (`dlf` for simple words, `dlc` for compound words and `err` for unknown words) are placed in the text dictionary. The `dlf` and `dlc` files are called text dictionaries.

As soon as the dictionary look-up is finished, Unitex displays the sorted lists of simple, compound and unknown words found in a new window. Figure 2.12 shows the result for an English text.

It is also possible to apply dictionaries without preprocessing the text. In order to do this, click on "Apply Lexical Resources..." in the "Text" menu. Unitex then opens a window (see figure 2.13) in which you can select the list of dictionaries to apply.

The list "User resources" lists all compressed dictionaries present in the directory `(current language)/Dela` of the user. The dictionaries installed in the system are listed in the scroll list named "System resources". Use the combination <Ctrl+click> to select multiple dictionaries. The button "Set Default" allows you to define the current selection of dictionaries as the default. This default selection will then be used during preprocessing if you activate the option "Apply All default Dictionaries". If you right-click on a dictionary name, the associated documentation, if any, will be displayed in the lower frame of the window.

2.5.6 Analysis of compound words in German, Norwegian and Russian

In certain languages like Norwegian, German and others, it is possible to form new compound words by concatenating together other words. For example, the word *aftenblad* meaning *evening journal* is obtained by combining the words *aften* (*evening*) et *blad* (*journal*). The program `PolyLex` searches the list of unknown words after the application of dictionaries and tries to treat each of these words as a compound word. If a word has at least one analysis as a compound word, it is deleted from the list of unknown words and the lines produced for this word are appended to the text dictionary of simple words.

2.6 Opening a tagged text

A tagged text is a text containing words with lexical tags enclosed in round brackets:

I do not like the {square bracket,.N} sign! {S}

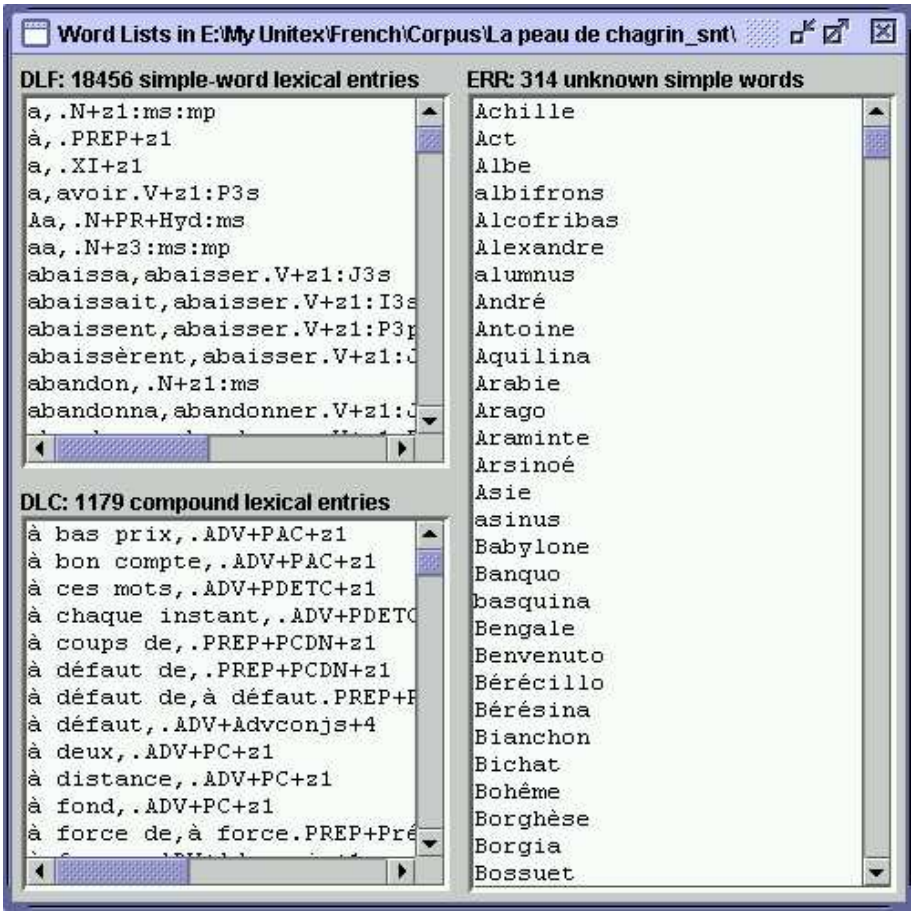


Figure 2.12: Result after applying dictionaries to a French text

Such tags can be used to avoid ambiguities. In the previous example, it will be impossible to match *square bracket* as the combination of two simple words.

However, the presence of these tags can alter the application of preprocessing graphs. To avoid complications, you can use the "Open Tagged Text..." command in the "Text" menu. With it, you can open a tagged text and skip the application of preprocessing graphs, as shown on Figure 2.14.

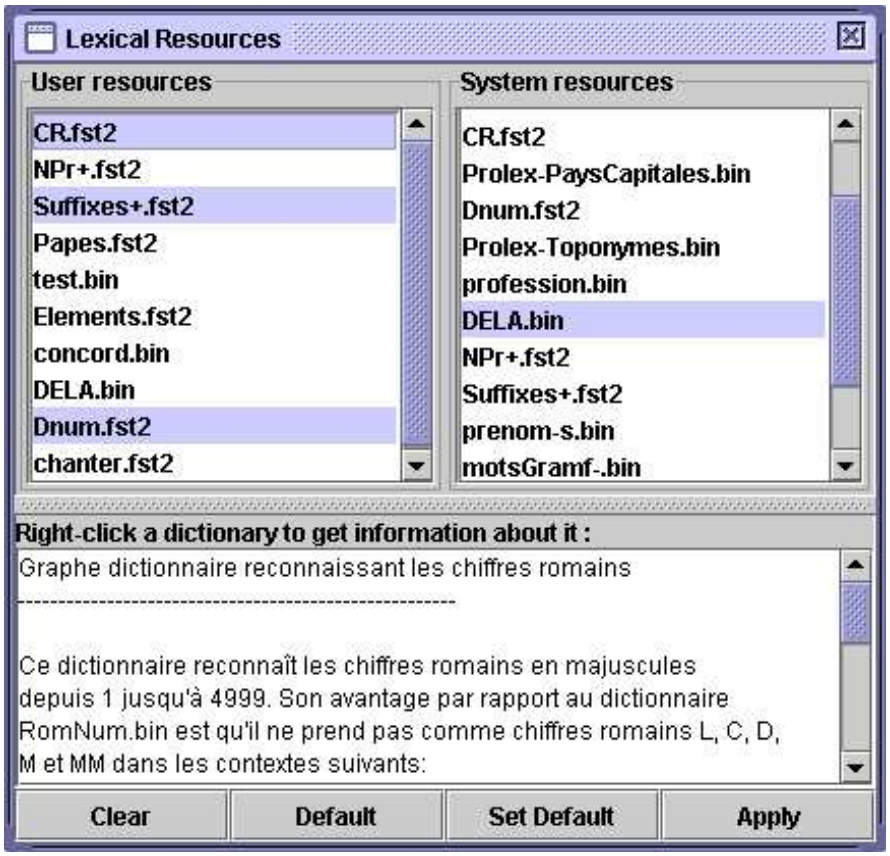


Figure 2.13: Parameterizing the application of dictionaries

Figure 2.14: Preprocessing a tagged text

Chapter 3

Dictionaries

3.1 The DELA dictionaries

The electronic dictionaries distributed with Unitex use the DELA syntax (Dictionnaires Electroniques du LADL, LADL electronic dictionaries). This syntax describes the simple and compound lexical entries of a language with their grammatical, semantic and inflectional information. We distinguish two kinds of electronic dictionaries. The one that is used most often is the dictionary of inflected forms DELAF (DELA de formes Fléchies, DELA of inflected forms) or DELACF (DELA de formes Composées Fléchies, DELA of compound inflected forms) in the case of compound forms. The second type is a dictionary of non-inflected forms called DELAS (DELA de formes simples, simple forms DELA) or DELAC (DELA de formes composées, compound forms DELA).

Unitex programs make no distinction between simple and compound form dictionaries. We will use the terms DELAF and DELAS to distinguish the two kinds of dictionaries whose entries are simple, compound, or mixed forms.

3.1.1 The DELAF format

Entry syntax

An entry of a DELAF is a line of text terminated by a newline that conforms to the following syntax:

```
apples,apple.N+conc:p/this is an example
```

The different elements of this line are:

- `apples` is the inflected form of the entry; it is mandatory;
- `apple` is the canonical form of the entry. For nouns and adjectives (in French), it is usually the masculine singular form; for verbs, it is the infinitive. This information may be left out as in the following example:

```
apple,.N+conc:s
```

This means that the canonical form is the same as the inflected form. The canonical form is separated from the inflected form by a comma.

- `N+conc` is the sequence of grammatical and semantic information. In our example, `N` designates a noun, and `conc` indicates that this noun designates a concrete object (see table 3.2).

  Each entry must have at least one grammatical or semantic code, separated from the canonical form by a period. If there are more codes, these are separated by the `+` character.

- `:p` is an inflectional code which indicates that the noun is plural. Inflectional codes are used to describe gender, number, declination, and conjugation. This information is optional. An inflectional code is made up of one or more characters that represent one information each. Inflectional codes have to be separated by the `:` character, for instance in an entry like the following:

  ```
  hang,.V:W:P1s:P2s:P1p:P2p:P3p
  ```

  The `:` character is interpreted in OR semantics. Thus, `:W:P1s:P2s:P1p:P2p:P3p` means "infinitive", "1st person singular present", "2nd person singular present", etc. (see table 3.3) Since each character represents one information, it is not necessary to use the same character more than once. In this way, encoding the past participle using the code `:PP` would be exactly equivalent to using `:P` alone;

- `/this is an example` is a comment. Comments are optional and are introduced by the `/` character. These comments are left out when the dictionaries are compressed.

IMPORTANT REMARK: It is possible to use the full stop and the comma within a dictionary entry. In order to do this they have to be escaped using the `\` character:

```
1\,000,one thousand.NUMBER
United Nations,U\.N\..ACRONYM
```

ATTENTION: Each character is taken into account within a dictionary line. For example, if you insert spaces, they are considered to be a part of the information. In the following line:

```
hath,have.V:P3s /old form of 'has'
```

The space that precedes the / character will be considered to be part of a 4-character inflectional code.

It is possible to insert comments into a DELAF or DELAS dictionary by starting the line with a / character. Example:

```
/ 'English' designates a pool spin
English,.N+z3:s
```

Compound words with spaces or dashes

Certain compound words like *acorn-shell* can be written using spaces or dashes. In order to avoid duplicating the entries, it is possible to use the = character. At the time when the dictionary is compressed, the `Compress` program verifies for each line if the inflected or canonical form contains a non-escaped = character. If this is the case, the program replaces this by two entries: one where the = character is replaced by a space, and one where it is replaced by a dash. Thus, the following entry:

```
acorn=shells,acorn=shell.N:p
```

is replaced by the following entries:

```
acorn shells,acorn shell.N:p
acorn-shells,acorn-shell.N:p
```

NOTE: If you want to keep an entry that includes the = character, escape it using \ as in the following example:

```
E\=mc2,.FORMULA
```

This replacement is done when the dictionary is compressed. In the compressed dictionary, the escaped = characters are replaced by simple =. As such, if a dictionary containing the following lines is compressed:

```
E\=mc2,.FORMULA

acorn=shell,.N:s
```

and if the dictionary is applied to the following text:

Formulas like E=mc2 have nothing to do with acorn-shells.

you will get the following lines in the dictionary of compound words of the text:

```
E=mc2,.FORMULA

acorn-shells,.N:p
```

Entry Factorization

Several entries containing the same inflectional and canonical forms can be combined into a single one if they have the same grammatical and semantic codes. Among other things this allows us to combine identical conjugations for a verb:

```
bottle,.V:W:P1s:P2s:P1p:P2p:P3p
```

If the grammatical and semantic information differ, one has to create distinct entries:

```
bottle,.N+Conc:s
bottle,.V:W:P1s:P2s:P1p:P2p:P3p
```

Certain entries that have the same grammatical and semantic entries can have different senses, as it is the case for the French word *poêle* that describes a stove or a type of sheet in the masculine sense and a kitchen instrument in the feminine sense. You can thus distinguish the entries in this case:

```
poêle,.N+z1:fs/ poêle à frire
poêle,.N+z1:ms/ voile, linceul; appareil de chauffage
```

NOTE: In practice this distinction has the only consequence that the number of entries in the dictionary increases.

In the different programs that make up Unitex these entries are reduced to

```
poêle,.N+z1:fs:ms
```

Whether this distinction is made is thus left to the maintainers of the dictionaries.

3.1.2 The DELAS Format

The DELAS format is very similar to the one used in the DELAF. The only difference is that there is only one canonical form followed by grammatical and/or semantic codes. The canonical form is separated from the different codes by a comma. There is an example:

```
horse,N4+Anl
```

The first grammatical or semantic code will be interpreted by the inflection program as the name of the grammar used to inflect the entry. The entry of the example above indicates that the word *horse* has to be inflected using the grammar named `N4`. It is possible to add inflectional codes to the entries, but the nature of the inflection operation limits the usefulness of this possibility. For more details see below in section 3.4.

3.1.3 Dictionary Contents

The dictionaries provided with Unitex contain descriptions of simple and compound words. These descriptions indicate the grammatical category of each entry, optionally their inflectional codes, and diverse semantic information. The following tables give an overview of some of the different codes used in the Unitex dictionaries. These codes are the same for almost all languages, though some of them are special for certain languages (*i.e.* code for neuter nouns, etc.).

Code	Description	Examples
A	adjective	fabulous, broken-down
ADV	adverb	actually, years ago
CONJC	coordinating conjunction	but
CONJS	subordinating conjunction	because
DET	determiner	each
INTJ	interjection	eureka
N	noun	evidence, group theory
PREP	preposition	without
PRO	pronoun	you
V	verb	overeat, plug-and-play

Table 3.1: Frequent grammatical codes

Code	Description	Example
z1	general language	joke
z2	specialized language	floppy disk
z3	very specialized language	serialization
Abst	abstract	patricide
Anl	animal	horse
AnlColl	collective animal	flock
Conc	concrete	chair
ConcColl	collective concrete	rubble
Hum	human	teacher
HumColl	collective human	parliament
t	transitive verb	kill
i	intransitive verb	agree

Table 3.2: Some semantic codes

NOTE: The descriptions of tense in table 3.3 correspond to French. Nontheless, the majority of these definitions can be found in other languages (infinitive, present, past participle, etc.).

In spite of a common base in the majority of languages, the dictionaries contain encoding particularities that are specific for each language. Thus, as the declination codes vary a lot between different languages, they are not described here. For a complete description of all codes used within a dictionary, we recommend that you contact the author of the dictionary directly.

Code	Description
m	masculine
f	feminin
n	neuter
s	singular
p	plural
1, 2, 3	1st, 2nd, 3rd person
P	present indicative
I	imperfect indicative
S	present subjunctive
T	imperfect subjunctive
Y	present imperative
C	present conditional
J	simple past indicative
W	infinitive
G	present participle
K	past participle
F	future indicative

Table 3.3: Common inflectional codes

However, these codes are not exclusive. A user can introduce codes himself and can create his own dictionaries. For example, for educational purposes one could use a marker "faux-ami" in a French dictionary:

```
blesser,.V+faux-ami/injure
casque,.N+faux-ami/helmet
journée,.N+faux-ami/day
```

It is equally possible to use dictionaries to add extra information. Thus, you can use the inflected form of an entry to describe an abbreviation and the canonical form to provide the complete form:

```
DNA,DeoxyriboNucleic Acid.ACRONYM
LADL,Laboratoire d'Automatique Documentaire et Linguistique.ACRONYM
UN,United Nations.ACRONYM
```

3.2 Verfication of the dictionary format

When dictionaries become larger, it becomes tiresome to verify them by hand. Unitex contains the program `CheckDic` that automatically verifies the format of DELAF and DELAS dictionaries.

This program verifies the syntax of the entries. For each malformed entry the program outputs the line number, the contents of the line and the type of error. The results are saved in the file `CHECK_DIC.TXT` which is displayed when the verification is finished. In addition to eventual error messages, the file also contains the list of all characters used in the inflectional and canonical forms, the list of grammatical and semantic codes, and the list of inflectional codes used. The character list makes it possible to verify that the characters used in the dictionary are consistent with those in the alphabet file of the language. Each character is followed by its value in hexadecimal notation.

These code lists can be used to verify that there are no typing errors in the codes of the dictionary.

The program works with non-compressed dictionaries, i.e. the files in text format. The general convention is to use the `.dic` extension for these dictionaries. In order to verify the format of a dictionary, you first open it by choosing "Open..." in the "DELA" menu.

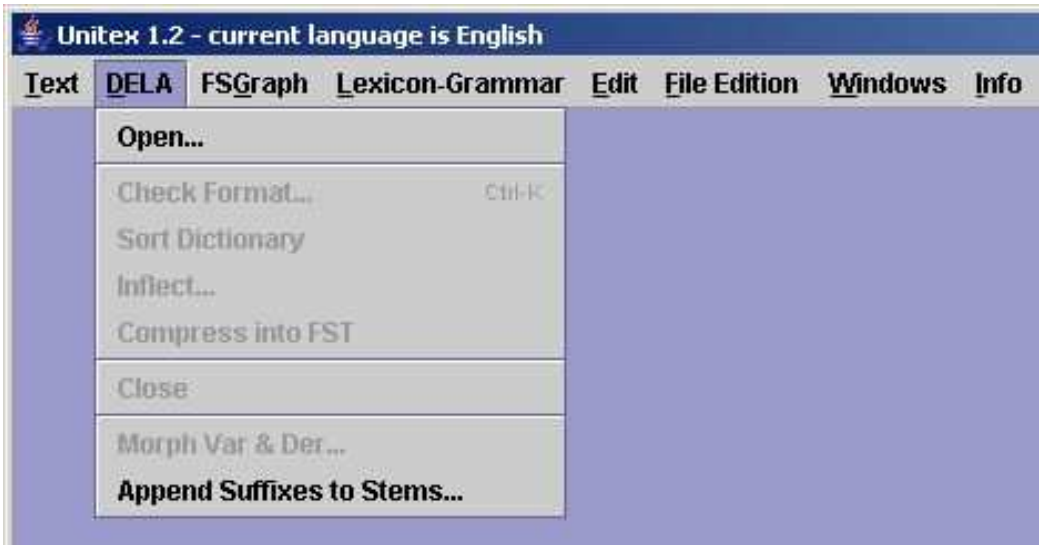


Figure 3.1: "DELA" Menu

Let's load the dictionary as in figure 3.2:

In order to start the automatic verification, click on "Check Format..." in the "DELA" menu. A window like in figure 3.3 is opened:

In this window you choose the dictionary type you want to verify. The results of verifying the dictionary in figure 3.2. are shown in figure 3.4.

The first error is caused by a missing period. The second, by the fact that no comma was found after the end of an inflected form. The third error indicates that the program didn't find any grammatical or semantic codes.

Figure 3.2: Dictionary example

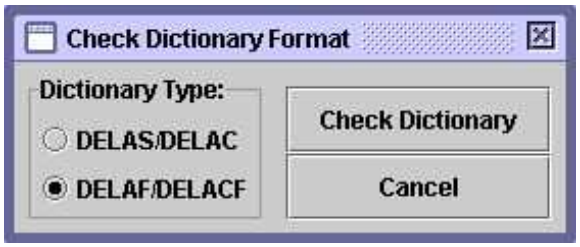


Figure 3.3: Automatic verification of a dictionary

3.3 Sorting

Unitex uses the dictionaries without having to worry about the order of the entries. When displaying them it is sometimes preferable to sort the dictionaries. The sorting depends on a number of criteria, first of all on the language of the text. Therefore the sorting of a Thai dictionary is done according to an order different from the alphabetical order. So different in fact that Unitex uses a sorting procedure developed specifically for Thai (see chapter 9).

For European languages the sorting is usually done in terms of the lexicographical order, although there are some variants. Certain languages like French treat some characters as equivalent. For example, the difference between the characters `e` and `é` is ignored if one wants to compare the words `manger` et `mangés` because the contexts r and s allow to decide the order. The difference is only taken into account when the contexts are identical, as they are when comparing `pêche` and `pèche`.

To allow for such effect, the sort program `SortTxt` uses a file which defines the equivalence of characters. This file is named `Alphabet_sort.txt` and can be found in the user directory for the current language. By default the first lines of this file for French look like this:

```
AÀÂÄaàâä
Bb
CÇcç
Dd
```

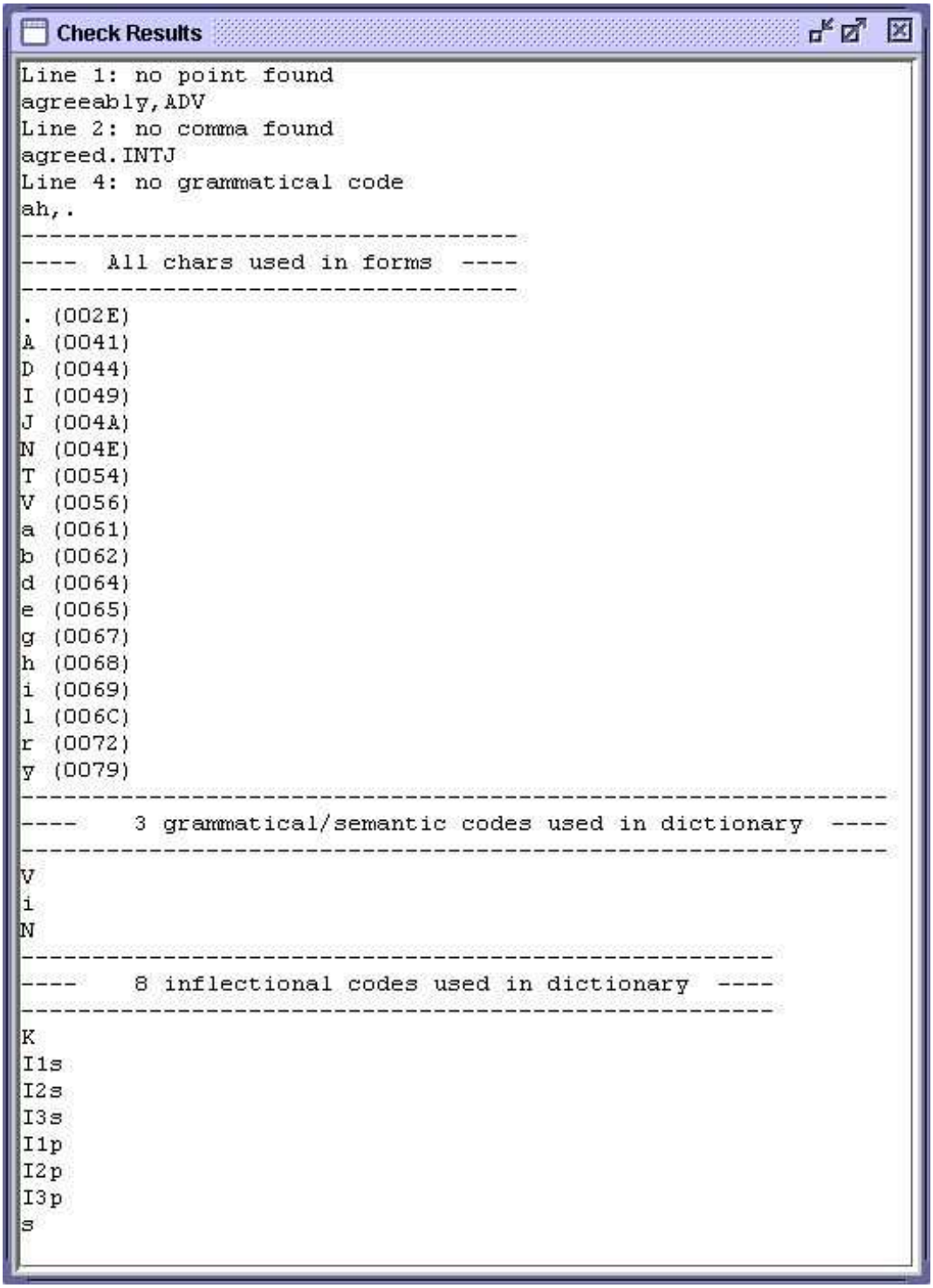


Figure 3.4: Results of the automatic verification

EÉÈÊËeéèêë

Characters in the same line are considered equivalent if the context permits. If two equivalent characters must be compared, they are sorted in the order they appear in from left to right. As can be seen from the extract above, there is no difference between lower and upper case. Accents and the cédille character are ignored as well.

To sort a dictionary, open it and then click on "Sort Dictionary" in the DELA menu. By default, the program always looks for the file `Alphabet_sort.txt` . If that file doesn't exist, the sorting is done according to the character indices in the Unicode encoding. By modifying that file, you can define your own sorting order.

Remark: After applying the dictionaries to a text, the files `dlf`, `dlc` and `err` are automatically sorted using this program.

3.4 Automatic inflection

As described in section 3.1.2, a line in a DELAS consists of a canonical form and a sequence of grammatical or semantic codes:

```
aviatrix,N4+Hum
matrix,N4+Math
radix,N4
```

The first code is interpreted as the name of the grammar used to inflect the canonical form. These inflectional grammars have to be compiled (see chapter 5). In the example above, all entries will be inflected by a grammar named `N4`.

In order to inflect a dictionary, click on "Inflect..." in the "DELA" menu. The window in figure 3.5 allows you to specify the directory in which inflectional grammars are found. By default, the subdirectory `Inflection` of the directory for the current language is used. The option "Add : before inflectional code if necessary" automatically inserts a ':' character before the inflectional codes if those codes do not start with this character. The option "Remove class numbers" is used to replace codes with the numbers used in the DELAS by codes without numbers. Example: `V17` will be replaced by `V` and `N4+Hum` by `V` and `N+Hum`.

Figure 3.6 shows an example of an inflectional grammar.

The paths describe the suffixes to add or to remove to get to an inflected form from a canonical form, and the outputs (text in bold under the boxes) are the inflectional codes to add to a dictionary entry.

In our example, two paths are possible. The first doesn't modify the canonical form and adds the inflectional code `:s`. The second deletes a letter with the `L` operator, then adds the `ux` suffix and adds the inflectional code `:mp`. Four operators are possible:

- `L` (left) remove a letter from the entry

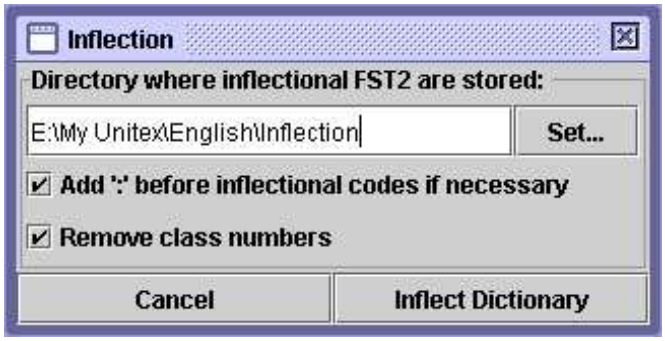


Figure 3.5: Configuration of automatic inflection

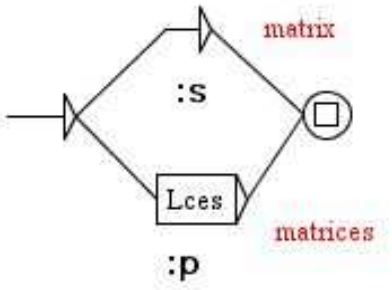


Figure 3.6: Inflectional grammar `N4`

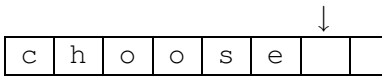
- `R` (right) restore a letter to the entry. In French, many verbs of the first group are conjugated in the present singular of the third person form by removing the `r` of the infinitive and changing the 4th letter from the end to `è`: `peler` → `pèle`, `acheter` → `achète`, `gérer` → `gère`, etc. Instead of describing an inflectional suffix for each verb (`LLLLèle`, `LLLLète` et `LLLLère`), the `R` operator can be used to describe it in one way: `LLLLèRR`.

- `C` (copy) duplicates a letter in the entry and moves everything on its right by one position. In cases like `permitted` or`hopped`, we see a duplication of the final consonant of the verb. To avoid writing an inflectional graph for every possible final consonant, one can use the `C` operator to duplicate any final consonant.

- `D` (delete) deletes a letter, shifting anything located on the right of this letter. For instance, if you want to inflect the Romanian word `european` into `europeni`, you must use the sequence `LDRi`. `L` will move the cursor on the `a`, `D` will delete the `a`, shifting the `n` on the left, and then `Ri` will restore the `n` and add an `i`.

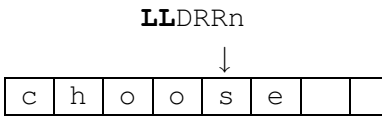
In the example below the inflection of `choose` is shown. The sequence `LLDRRn` describes the form `chosen`:

- Step 0: the canonical form is copied on the stack, and the cursor is set behind the last letter:

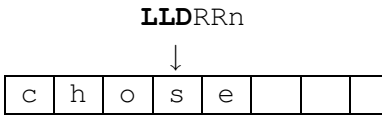


- Step 1: the cursor is moved one position to the left:

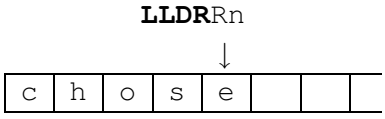
- Step 2: the cursor is moved one position to the left again:



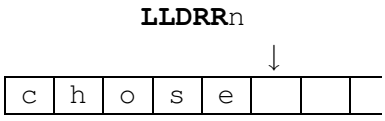
- Step 3: one character is deleted; everything to the right of the cursor is shifted one position to the left:



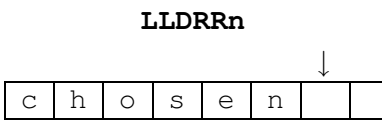
- Step 4: the cursor is moved to the right:



- Step 5: and to the right again:



- Step 6: the character `n` is pushed on the stack:



When all operations have been fulfilled, the inflected form consists of all letters to the right of the cursor (here `chosen`).

The inflection program `Inflect` traverses all paths of the inflectional grammar and tries all possible forms. In order to avoid having to replace the names of inflectional grammars by the real grammatical codes in the dictionary used, the program replaces these names by the longest prefixes made of letters if you have selected the "Remove class numbers" button. Thus, `N4` is replaced by `N`. By choosing the inflectional grammar names carefully, one can construct a ready to use dictionary.

Let's have a look at the dictionary we get after the DELAS inflection in our example:

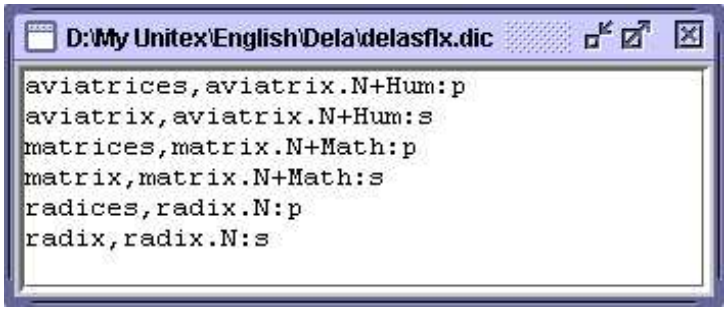


Figure 3.7: Result of automatic inflection

3.5 Compression

Unitex applies compressed dictionaries to the text. The compression reduces the size of the dictionaries and speeds up the lookup. This operation is done by the `Compress` program.
This program takes a dictionary in text form as input (for example `my_dico.dic`) and produces two files:

- `my_dico.bin` contains the minimal automaton of the inflected forms of the dictionaries;
- `my_dico.inf` contains the codes extracted from the original dictionary.

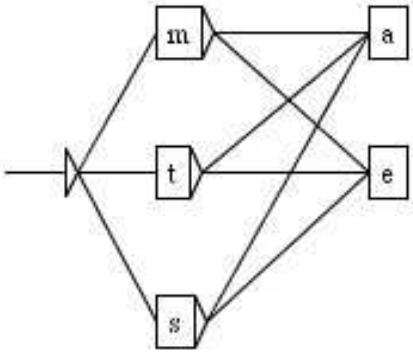


Figure 3.8: Representation of an example of a minimal automaton

The minimal automaton in the `my_dico.bin` file is a representation of inflected forms in which all common prefixes and suffixes are factorized. For example, the minimal automaton of the words `me`, `te`, `se`, `ma`, `ta` et `sa` can be represented by the graph in figure 3.8.

To compress a dictionary, open it and click on "Compress into FST" in the "DELA" menu. The compression is independent from the language and from the content of the dictionary. The messages produced by the program are displayed in a window that is not closed automatically. You can see the size of the resulting `.bin` file, the number of lines read and the number of inflectional codes created. Figure 3.9 shows the result of the compression of a dictionary of simple words.

The resulting files are compressed to about 95% for dictionaries containing simple words and 50% for those with compound words.

3.6 Applying dictionaries

Dictionaries can be applied (1) after pre-processing or (2) by explicitly clicking on "Apply Lexical Resources" in the "Text" menu (see section 3.6).

Unitex can manipulates compressed dictionaries (`.bin`) and dictionary graphs (`.fst2`). We will now describe the rules for applying dictionaries in detail. Dictionary graphs will be described in section 3.6.3.

3.6.1 Priorities

The priority rule says that if a word in a text is found in a dictionary, this word will not be taken into account by dictionaries with lower priority.

This allows for eliminating a part of ambiguity when applying dictionaries. For example, the French word *par* has a nominal interpretation in the golf domain. If you don't want to use this reading, it is sufficient to create a filter dictionary containing only the entry `par,.PREP`

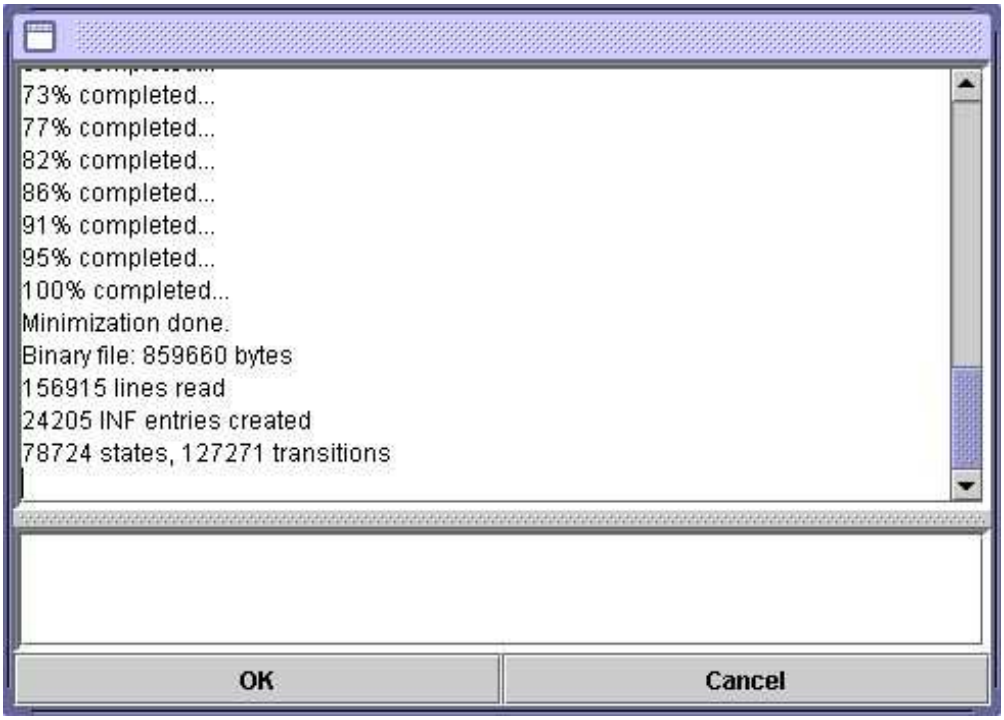


Figure 3.9: Results of a compression

and to apply this with highest priority. This way, even if dictionaries of simple words contain a different entry, this will be ignored given the priority rule.

There are three priority levels. The dictionaries whose names without extension end with - have the highest priority; those that end with + have the lowest one. All other dictionaries are applied with medium priority. The order in which dictionaries with the same priority are applied does not matter. On the command line, the command

```
Dico ex.snt alph.txt countries+.bin cities-.bin rivers.bin regions-.bin
```

will apply the dictionaries in the following order (`ex.snt` is the text to which the dictionaries are applied, and `alph.txt` is the alphabet file used):

1. `cities-.bin`
2. `regions-.bin`
3. `rivers.bin`
4. `countries+.bin`

3.6.2 Application rules for dictionaries

Besides the priority rule, the application of dictionaries respects upper case letters and spaces. The upper case rule is as follows:

- if there is an upper case letter in the dictionary, then an upper case letter has to be in the text;
- if a lower case letter is in the dictionary, there can be either an upper or lower case letter in the text.

Thus, the entry `peter,.N:fs` will match the words `peter`, `Peter` et `PETER`, while `Peter,.N+firstName` only recognizes `Peter` and `PETER`. Lower and upper case letters are defined in the alphabet file passed to the `Dico` program as a parameter.

Respecting white space is a very simple rule: For each sequence in the text to be recognized by a dictionary entry, it has to have exactly the same number of spaces. For example, if the dictionary contains `aujourd'hui,.ADV`, the sequence `Aujourd' hui` will not be recognized because of the space that follows the apostrophe.

3.6.3 Dictionary graphs

The `Dico` program can apply dictionary graphs. Dictionary graphs conform to the following rule: if applied by `Locate` in MERGE mode, they must produce output sequences that are valid DELAF lines. Figure 3.10 shows a graph that recognizes chemical elements. We can observe a first advantage of graphs over usual dictionaries: we can force case with double quotes. Thus, this graph will correctly match `Fe` but not `FE`, while this restriction cannot be specified in a normal DELAF.

Since dictionary graphs are applied using the engine of `Locate`, they have exactly the same properties than syntactic graphs. In particular character sequences below the token level can't be matched.

Another advantage of dictionary graphs is that they can use results given by previous dictionaries. Thus, it is possible to apply the standard dictionary, and then tag as proper names all the unknown words that begin with an uppercase letter, thanks to the graph `NPr+` shown in figure 3.11. The `+` in the graph name gives to it a low priority, so that it will be applied after the standard dictionary. This graph works with words that are still unknown after the application of the standard dictionary. Square brackets stand for a context definition. For more information about contexts, see section 6.3.

As dictionary graphs are applied by the `Locate` engine, you can use in them anything that `Locate` tolerates. In particular, you can use morphological filters. For instance, the graph shown on Figure 3.12 use such filters to recognize roman numerals. Note that it also uses contexts in order to avoid recognizing uppercase letters in some contexts.

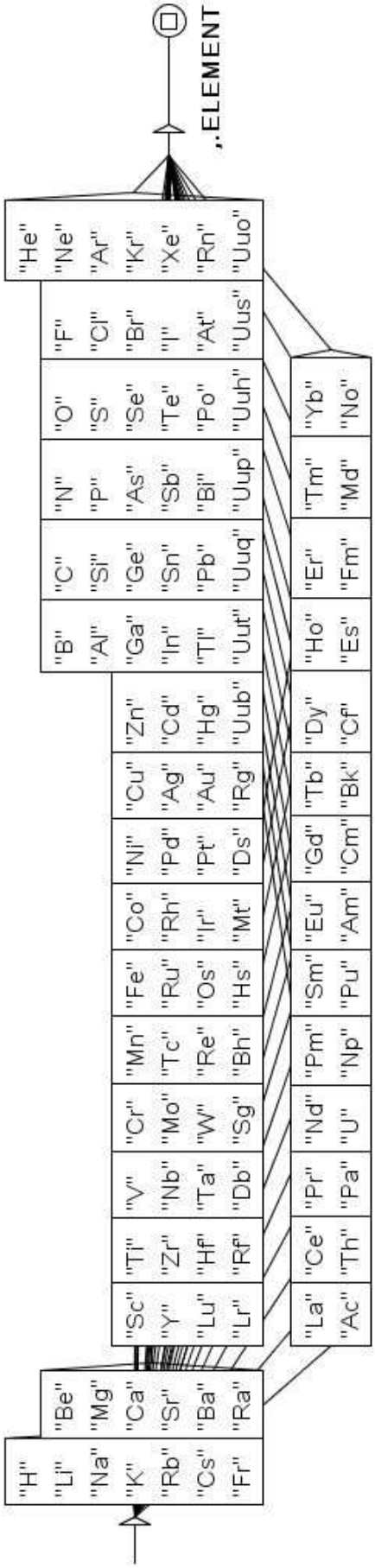


Figure 3.10: Dictionary graph of chemical elements

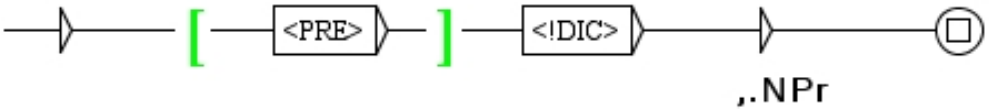


Figure 3.11: Dictionary graph that tags unknown words beginning with an uppercase letter as proper names

3.7 Bibliography

Table 3.4 gives some references for electronic dictionaries with simple and compound words. For more details, see the references page on the Unitex website:
http://www-igm.univ-mlv.fr/~unitex

Language	Simple words	Compound words
English	[30], [40]	[11], [45]
French	[14], [15], [34]	[15], [25], [46], [27]
Modern Greek	[2], [13], [31]	[32], [33]
Italian	[19], [20]	[50]
Spanish	[5]	[4]

Table 3.4: Some bibliographical references for electronic dictionaries

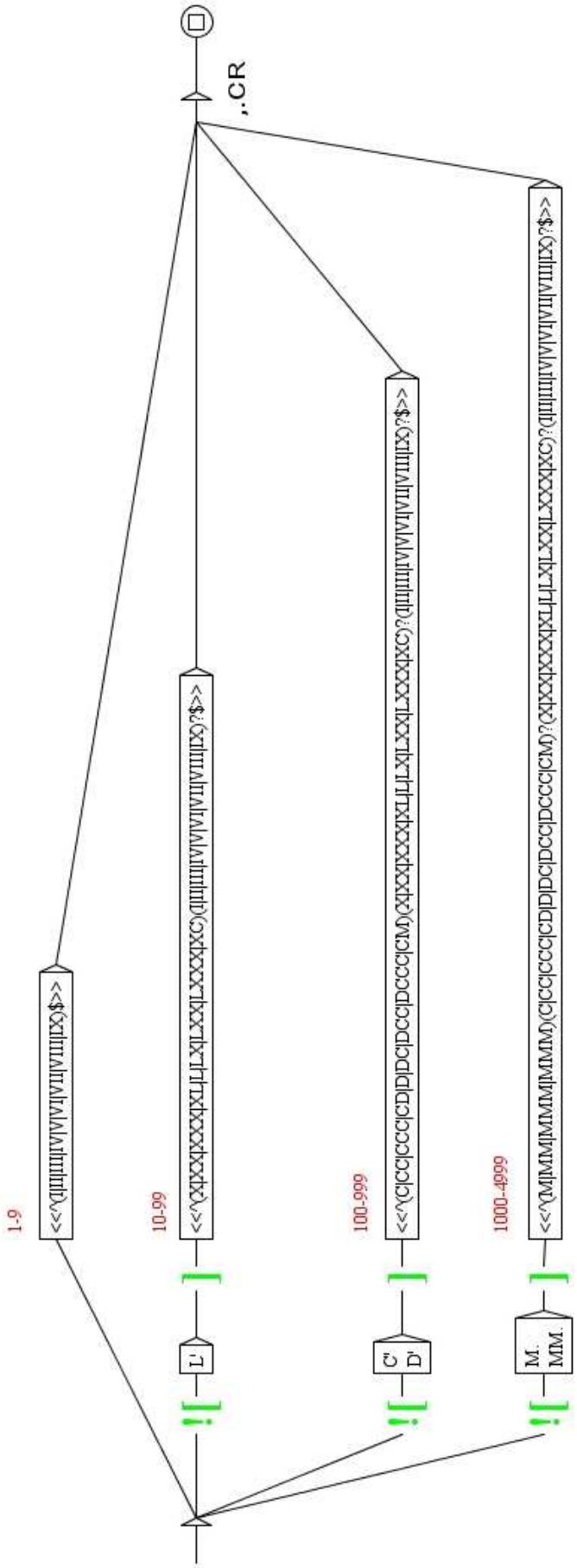


Figure 3.12: Dictionary graph of roman numerals

Chapter 4

Searching with regular expressions

This chapter describes how to search a text for simple patterns by using regular expressions.

4.1 Definition

The goal of this chapter is not to give an introduction on formal languages but to show how to use regular expressions in Unitex in order to search for simple patterns. Readers who are interested in a more formal presentation can consult the many works that discuss regular expression patterns.

A regular expression can be:

- a token (`book`) or a lexical mask (`<smoke.V>`);
- the concatenation of two regular expressions (`he smokes`);
- the union of two regular expressions (`Pierre+Paul`);
- the Kleene star of a regular expression (`bye*`).

4.2 Tokens

In a regular expression, a token is defined as in 2.5.4 (page 24). Note that the symbols dot, plus, star, less than, opening and closing parentheses and double quotes have a special meaning. It is therefore necessary to precede them with an escape character `\` if you want to search for them. Here are some examples of valid tokens:

```
cat
\.
<N:ms>
{S}
```

By default Unitex is set up to let lower case patterns also find upper-case matches. It is possibe to enforce case-sensitive matching using quotation marks. Thus, `"peter"` recognizes only the form `peter` and not `Peter` or `PETER`.

NOTE: in order to make a space obligatory, it needs to be enclosed in quotation marks.

4.3 Lexical masks

A lexical mask is a search query that matches tokens or sequences of tokens.

4.3.1 Special symbols

There are two kinds of lexical masks. The first category contains all symbols that have been introduced in section 2.5.2 except for the symbol `<PNC>`, which matches punctuation signs, and `<^>`, which matches a line feed. Since all line feeds have been replaced by spaces this symbol cannot longer be useful when searching for lexical masks. These symbols, also called *meta-symbols*, are the following:

- <E> : the empty word or epsilon. Matches the empty string;
- <TOKEN> : matches any token, except the space; used by default for morphological filters
- <MOT> : matches any token that consists of letters;
- <MIN> : matches any lower-case token;
- <MAJ> : matches any upper-case token;
- <PRE> : matches any token that consists of letters and starts with a capital letter;
- <DIC> : matches any word that is present in the dictionaries of the text;
- <SDIC> : matches any simple word in the text dictionaries;
- <CDIC> : matches any composed word in the dictionaries of the text;
- <NB> : matches any contiguous sequence of digit (1234 is matched but not 1 234);
- `#` : prohibits the presence of space.

NOTE: as described in section 2.5.4, NO meta can be used to match the `{STOP}` marker, not even <TOKEN>.

4.3.2 References to information in the dictionaries

The second kind of lexical masks refers to the information in the dictionaries of the text. The four possible forms are:

- `<be>`: matches all the entries that have `be` as canonical form;
- `<be.V>`: matches all entries having `be` as canonical form and the grammatical code `V`;
- `<V>`: matches all entries having the grammatical code `V`;
- `{am,be.V} or <am,be.V>`: matches all the entries having `am` as inflected form, `be` as canonical form and the grammatical code `V`. This kind of lexical mask is only of interest if applied to the text automaton where all the ambiguity of the words is explicit. While executing a search on the text, that lexical mask matches the same as the simple token `am`.

4.3.3 Grammatical and semantic constraints

The references to dictionary information (`be`, `V`) in these examples are elementary. It is possible to express more complex lexical masks by using several grammatical or semantic codes separated by the character `+`. An entry of the dictionary is then only found if it has all the codes that are present in the mask. The mask `<N+z1>` thus recognizes the entries:

```
broderies,broderie.N+z1:fp
capitales européennes,capitale européenne.N+NA+Conc+HumColl+z1:fp
```

but not:

```
Descartes,René Descartes.N+Hum+NPropre:ms
habitué,.A+z1:ms
```

It is possible to exclude codes by preceding them with the character – instead of `+`. In order to be recognized an entry has to contain all the codes authorized by the lexical mask and none of the prohibited codes. The mask `<A-z3>` thus recognizes all the adjectives that do not have the code `z3` (cf. table 3.2). If you want to refer to a code containing the character – you have to escape this character by preceding it with a `\`. Thus, the mask `<N+faux\-ami>` could recognize all entries of the dictionaries containing the codes `N` and `faux-ami`.

The order in which the codes appear in the mask is not important. The three following patterns are equivalent:

```
<N-Hum+z1>
<z1+N-Hum>
<-Hum+z1+N>
```

NOTE: it is not possible to use a lexical mask that only has prohibited codes. `<-N>` and `<-A-z1>` are thus incorrect masks. However, you can express such constraints using contexts (see section 6.3).

4.3.4 Inflectional constraints

It is also possible to specify constraints about the inflectional codes. These constraints have to be preceded by at least one grammatical or semantic code. They are represented as inflectional codes present in the dictionaries. Here are some examples of lexical masks using inflectional constraints:

- `<A:m>` recognizes a masculine adjective;
- `<A:mp:f>` recognizes a masculine plural or a feminine adjective;
- `<V:2:3>` recognizes a verb in the 2nd or 3rd person; that excludes all tenses that have neither a 2nd or 3rd person (infinitive, past participle and present participle) as well as the tenses that are conjugated in the first person.

In order to let a dictionary entry E be recognized by mask M, it is necessary that at least one inflectional code of E contains all the characters of an inflectional code of M. Consider the following example:

E=`pretext,.V:W:P1s:P2s:P1p:P2p:P3p`
M=`<V:P3s:P3>`

No inflectional code of E contains the characters `P`, `3` and `s` at the same time. However, the code `P3p` of E does contain the characters `P` and `3`. The code `P3` is included in at least one code of E, mask M thus recognizes entry E. The order of the characters inside an inflectional code is without importance.

4.3.5 Negation of a lexical mask

It is possible to negate a lexical mask by placing the character `!` immediately after the character `<`. Negation is possible with the masks `<MOT>`, `<MIN>`, `<MAJ>`, `<PRE>`, `<DIC>` as well as with the masks that carry grammatical, semantic of inflectional codes (*i.e.* `<!V-z3:P3>`). The masks `#` and `" "` are the negation of each other. The mask `<!MOT>` recognizes all tokens that do not consist of letters except for the sentence separator `{S}` and the `{STOP}` marker. Negation has no effect on `<NB>`, `<SDIC>`, `<CDIC>` and `<TOKEN>`.

The negation is interpreted in a special way in the lexical masks `<!DIC>`, `<!MIN>`, `<!MAJ>` and `<!PRE>`. Instead of recognizing all forms that are not recognized by the mask without negation, these masks find only forms that are sequences of letters. Thus, the mask `<!DIC>` allows you to find all unknown words in a text. These unknown forms are mostly proper names, neologisms and spelling errors.

The negation of a dictionary mask like `<V:G>` will match any word, except for those that are matched by this mask. For instance, `<!V:G>` will not match the word `being`, even if there are homonymic non-verbal entries in the dictionaries:

```
being,.A
being,.N+Abst:s
being,.N+Hum:s
```

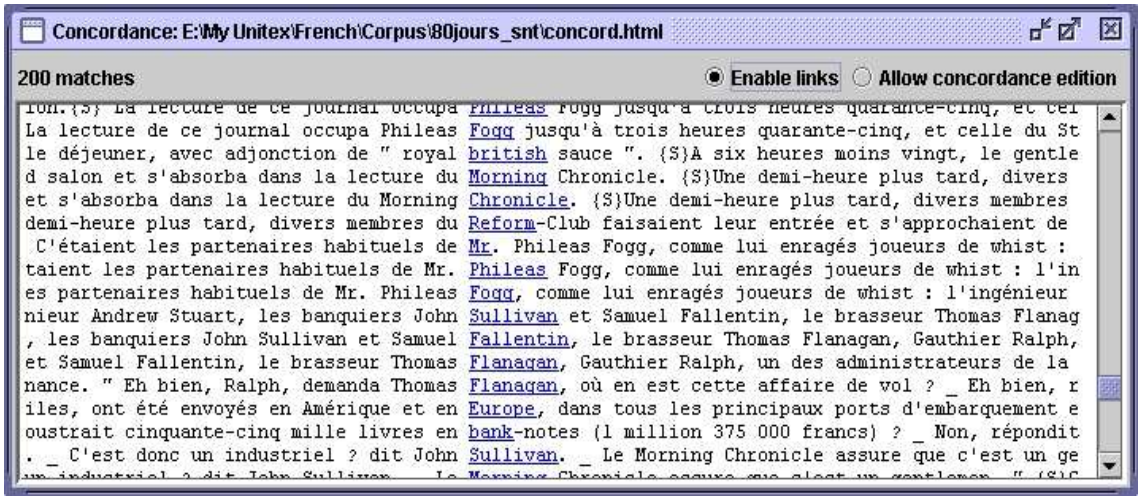


Figure 4.1: Result of the search for `<!DIC>`

Here are some examples of lexical masks with the different types of constraints:

- `<A-Hum:fs>` : a non-human adjective in the feminine singular;
- `<lire.V:P:F>` : the verb *lire* in the present or future tense;
- `<suis,suivre.V>` : the word *suis* as inflected form of the verb *suivre* (as opposed to the form of the verb *être*);
- `<facteur.N-Hum>` : all nominal entries that have *facteur* as canonical form and that do not have the semantic code `Hum`;
- `<!ADV>` : all words that are not adverbs;
- `<!MOT>` : all tokens that are not made of letters (cf. figure 4.2). This mask does not recognize the sentence separator `{S}` and the special tag `{STOP}`.

4.4 Concatenation

There are three ways to concatenate regular expressions. The first consists in using the concatenation operator which is represented by the period. Thus, the expression:

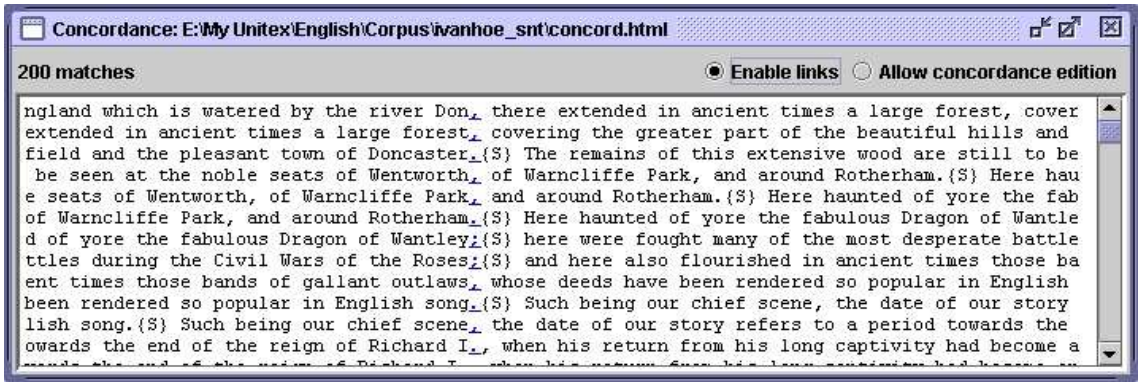


Figure 4.2: Result of a search for the pattern `<!MOT>`

```
<DET>.<N>
```

recognizes a determiner followed by a noun. The space can also be used for concatenation. The following expression:

```
the <A> cat
```

recognizes the token *the*, followed by an adjective and the token *cat*. Finally, it is possible to omit the dot and the space before an opening parenthesis or the character < as well as after a closing parenthesis or after the character >. The parenthesis are used as delimiters of a regular expression. All of the following expressions are equivalent:

```
the <A> cat
(the <A>)cat
the.<A>.cat
(the)<A>.cat
(the(<A>))(cat)
```

4.5 Union

The union of regular expressions is expressed by typing the character + between them. The expression

```
(I+you+he+she+it+we+they) <V>
```

recognizes a pronoun followed by a verb. If an element in an expression is optional it is sufficient to use the union of this element and the empty word epsilon.

Examples:

`the(little+<E>)cat` recognizes the sequences *the cat* and *the little cat*

`(<E>+Anglo-)(French+Indian)` recognizes *French*, *Indian*, *Anglo-French* and *Anglo-Indian*

4.6 Kleene star

The Kleene star, represented by the character `*`, allows you to recognize zero, one or several occurrences of an expression. The star must be placed on the right hand side of the element in question. The expression:

```
this is very* cold
```

recognizes *this is cold*, *this is very cold*, *this is very very cold*, etc. The star has a higher priority than the other operators. You have to use brackets in order to apply the star to a complex expression. The expression:

```
0,(0+1+2+3+4+5+6+7+8+9)*
```

recognizes a zero followed by a comma and by a possibly empty sequence of digits.

ATTENTION: It is prohibited to search for the empty word with a regular expression. If you try to search for `(0+1+2+3+4+5+6+7+8+9)*`, the program will flag an error as shown in figure 4.3.

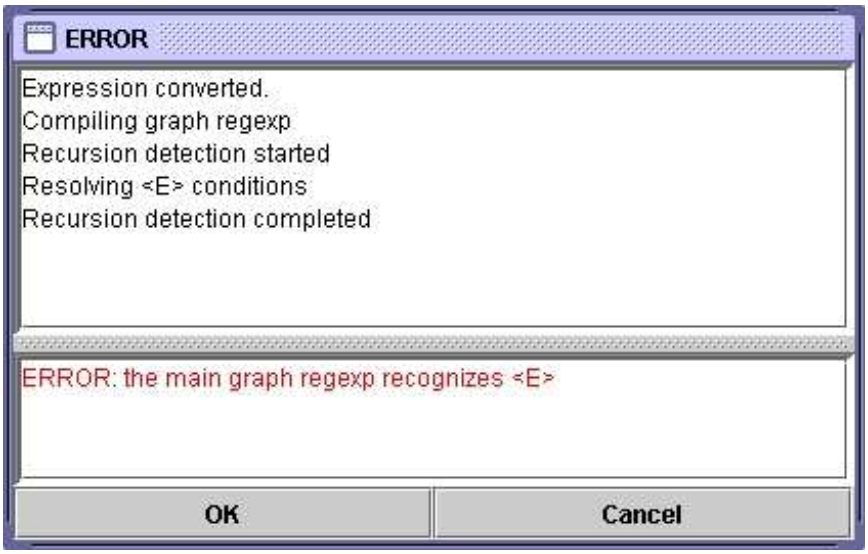


Figure 4.3: Error message when searching for the empty string

4.7 Morphological filters

It is possible to apply morphological filters to the lexemes found. For that, it is necessary to immediately follow the lexeme found by a filter in double angle brackets:

lexical mask<<morphological pattern>>

The morphological filters are expressed as regular expressions in POSIX format (see [36] for the detailed syntax). Here are some examples of elementary filters:

- `<<ss>>`: contains `ss`
- `<<^a>>`: begins with `a`
- `<<ez$>>`: ends with `ez`
- `<<a.s>>`: contains `a` followed by any character, followed by `s`
- `<<a.*s>>`: contains `a` followed by a sequence of any character, followed by `s`
- `<<ss|tt>>`: contains `ss` or `tt`
- `<<[aeiouy]>>`: contains a non accentuated vowel
- `<<[aeiouy]{3,5}>>`: contains a sequence of non-accentuated vowels whose length is between 3 and 5
- `<<ée?>>`: contains `é` followed by an optional `e`
- `<<ss[^e]?>>`: contains `ss` followed by an optional character which is not `e`

It is possible to combine these elementary filters to form more complex filters:

- `<<[ai]ble$>>`: ends with `able` or `ible`
- `<<^(anti|pro)-?>>`: begins with `anti` or `pro`, followed by an optional dash
- `<<^([rst][aeiouy]){2,}$>>`: a word formed by 2 or more sequences beginning with `r`, `s` or `t` followed by a non-accentuated vowel
- `<<^([^l]|l[^e])>>`: does not begin with `l` unless the second letter is an `e`, in other words, any word except the ones starting with `le`. Such constraints are better described using contexts (see section 6.3).

By default, a morphological filter alone is regarded as applying it to the lexical mask `<TOKEN>`, that means any token except space and `{STOP}`. On the other hand, when a filter follows a lexical mask immediately, it applies to what was recognized by the lexical mask. Here are some examples of such combinations:

- `<V:K><<i$>>`: Past participle ending with `i`
- `<CDIC><<->>`: A compound word containing a dash
- `<CDIC><< .* >>`: a compound word containing at least two spaces

- `<A:fs><<^pro>>`: a feminine singular adjective beginning with `pro`
- `<DET><<^([^u]|(u[^n])|(un.+))>>`: a (French) determiner different from `un`
- `<!DIC><<es$>>`: a word which is not in the dictionary and which ends with `es`
- `<V:S:T><<uiss>>`: a verb in the past or present subjunctive, and containing `uiss`

NOTE: By default, morphological filters are subject to the same variations of case as lexical masks. Thus, the filter `<<^é>>` will recognize all the words starting with `é`, but also those which start with `e` or `E`. To force the matcher to match case sensitive, add `_f_` immediately after the filter, e.g.: `<A><<^é>>_f_`.

4.8 Search

4.8.1 Configuration of the search

In order to search for an expression first open a text (cf. chapter 2). Then click on "Locate Pattern..." in the menu "Text". The window of figure 4.4 appears.

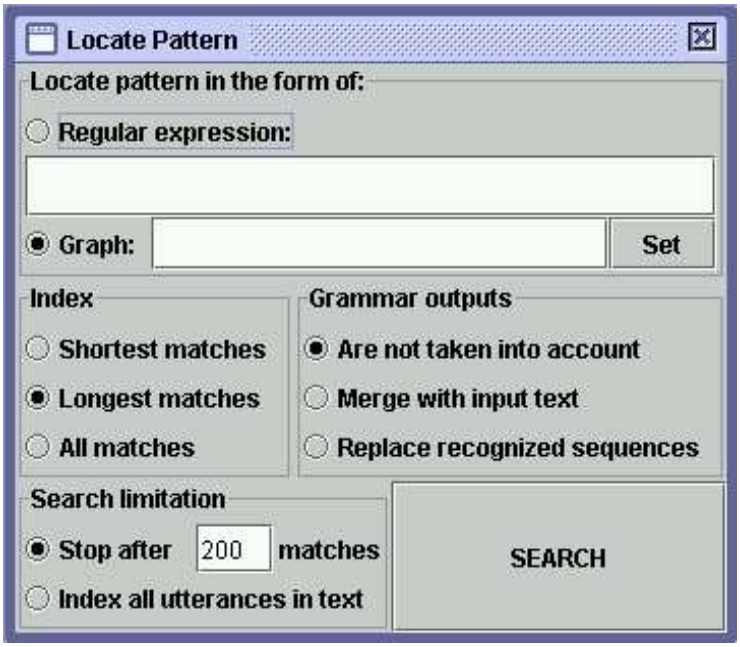


Figure 4.4: “Locate pattern” window

The box "Locate pattern in the form of" allows you to select regular expression or grammar. Click on "Regular expression".

The box "Index" allows you to select the recognition mode:

- "Shortest matches" : prefer short matches;
- "Longest matches" : prefer longer matches. This is the default;
- "All matches" : output all recognized sequences.

The box "Search limitation" is used to limit the number of results to a certain number of occurrences. By default, the search is limited to the first 200 occurrences.

The options of the box "Grammar outputs" do not concern regular expressions. They are described in section 6.7.

Enter an expression and click on "Search" in order to start the search. Unitex will transform the expression into a grammar in the `.grf` format. This grammar will then be compiled into a grammar of the `.fst2` format that will be used for the search.

4.8.2 Presentation of the results

When the search is finished, the window of figure 4.5 appears showing the number of matched occurrences, the number of recognized tokens and the ratio between this number and the total number of tokens in the text.

Figure 4.5: Search results

After having clicked on "OK" you will see window 4.6 appear, which allows you to configure the presentation of the matched occurrences. You can also open this window by clicking on "Display Located Sequences..." in the menu "Text". The list of occurrences is called a *concordance*.

The box "Modify text" offers the possibility to replace the matched occurrences with the generated outputs. This possibility will be examined in chapter 6.

The box "Extract units" allows you to create a text file with all the sentences that do or do not contain matched units. With the button "Set File", you can select the output file. Then

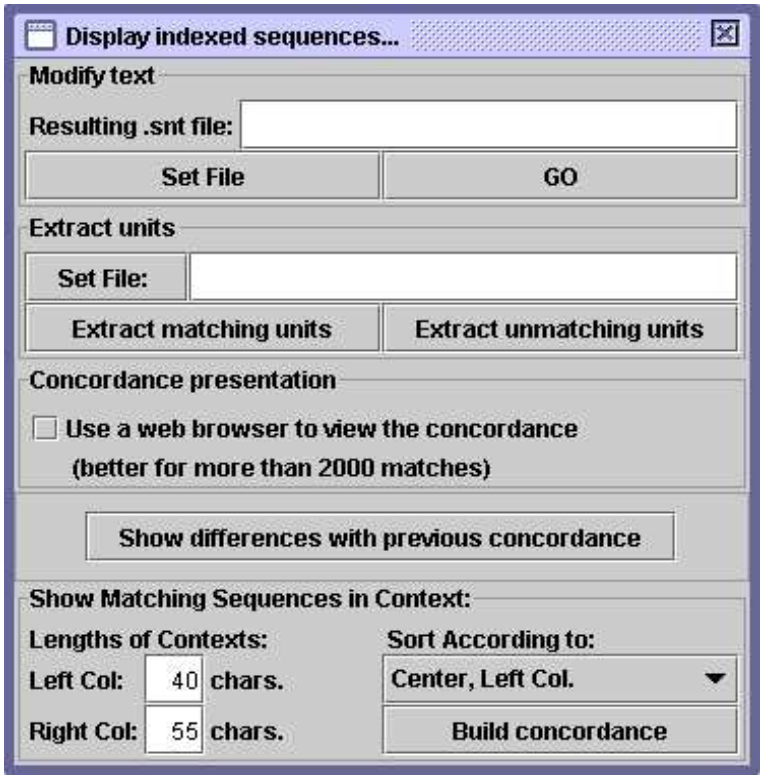


Figure 4.6: Configuration of the presentation of the found occurrences

click on "Extract matching units" or "Extract unmatching units" depending on whether you are interested in sentence with or without matching units.

In the box "Show Matching Sequences in Context" you can select the length in characters of the left and right contexts of the occurrences that will be presented in the concordance. If an occurrence has less characters than its right context the line will be completed with the necessary number of characters. If an occurrence has a length greater than that of the right context it will be displayed completely.

NOTE: in Thai, the size of the contexts is measured in displayable characters and not in real characters. This makes it possible to keep the line alignment in the concordance despite the presence of diacritics that combine with other letters instead of being displayed as normal characters.

You can choose the sort order in the list "Sort According to". The mode "Text Order" displays the occurrences in the order of their appearance in the text. The other six modes allow you to sort in columns. The three zones of a line are the left context, the occurrence and the right context. The occurrences and the right contexts are sorted from left to right. The left contexts are sorted from right to left. The default mode is "Center, Left Col.". The concordance is generated in the form of an HTML file.

If a concordance reaches several thousands of occurrences, it is advisable to display it in a web browser (Firefox [8], Netscape [9], Internet Explorer, etc.) instead. Check the box "Use a web browser to view the concordance" (cf. figure 4.6). This option is activated by default if the number of occurrences is greater than 3000. You can configure which web browser to use by clicking on "Preferences..." in the menu "Info". Click on the tab "Text Presentation" and select the program to use in the field "Html Viewer" (cf. figure 4.7).

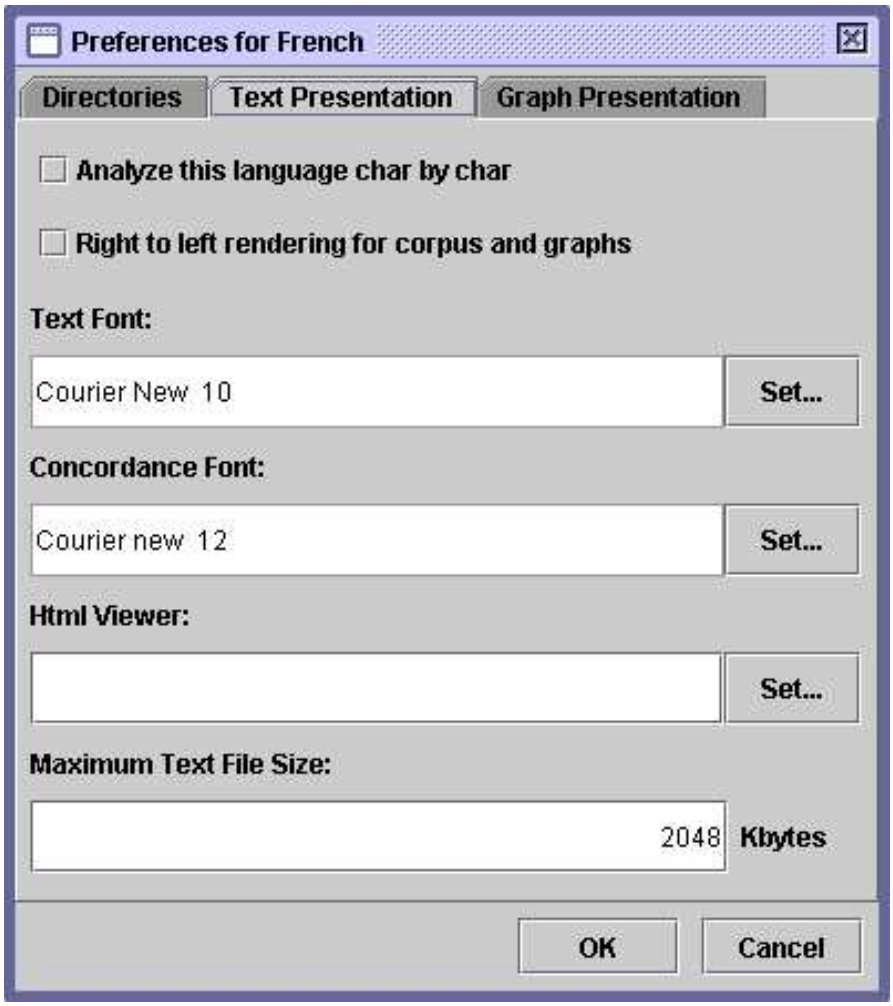


Figure 4.7: Selection of a web browser for displaying concordances

If you choose to open the concordance in Unitex, you will see a window as shown on Figure 4.8. The "Enable links" option, activated by default, makes utterances react as hyperlinks. If you click on an utterance, the text frame is opened and the corresponding sequence is highlighted. Moreover, if the text automaton is available and if this window is not iconified, the sentence automaton that contains the utterance will be shown. If you choose the

Figure 4.8: Example concordance

"Allow concordance edition" option, you can not use utterances as hyperlinks, but you can edit the concordance as text. In particular, you can navigate in the concordance with a cursor, which may be convenient when using large contexts.

Chapter 5

Local grammars

Local grammars are a powerful tool to represent the majority of linguistic phenomena. The first section presents the formalism in which these grammars are represented. Then we will see how to construct and present grammars using Unitex.

5.1 The local grammar formalism

5.1.1 Algebraic grammars

Unitex grammars are variants of algebraic grammars, also known as context-free grammars. An algebraic grammar consists of rewriting rules. Below you see a grammar that matches any number of a characters:

$S \rightarrow aS$
$S \rightarrow \varepsilon$

The symbols to the left of the rules are called *non-terminal symbols* since they can be replaced. Symbols that cannot be replaced by other rules are called *terminal symbols*. The items at the right side are sequences of non-terminal and terminal symbols. The epsilon symbol ε designates the empty word. In the grammar above, S is a non-terminal symbol and a a terminal (symbol). S can be rewritten as either an a followed by a S or as the empty word. The operation of rewriting by applying a rule is called *derivation*. We say that a grammar generates a word if there exists a sequence of derivations that produces that word. The non-terminal that is the starting point of the first derivation is called an *axiom*.

The grammar above also generates the word *aa*, since we can derive this word according to the axiom S by applying the following derivations:

Derivation 1: rewriting the axiom to aS
$\underline{S} \rightarrow aS$

Derivation 2: rewriting S at the right side of aS
$S \rightarrow a\underline{S} \rightarrow aaS$

Derivation 3: rewriting S to ε
$S \rightarrow aS \rightarrow aa\underline{S} \rightarrow aa$

We call the set of words generated by a grammar the *language generated by the grammar*. The languages generated by algebraic grammars are called *algebraic languages* or *context-free languages*.

5.1.2 Extended algebraic grammars

Extended algebraic grammars are algebraic grammars where the members on the right side of the rule are not just sequences of symbols but regular expressions. Thus, the grammar that generates a sequence of an arbitrary number of a's can be written as a grammar consisting of one rule:

$S \rightarrow a^*$

These grammars, also called *recursive transition networks* (*RTN*) or *syntax diagrams*, are suited for a user-friendly graphical representation. Indeed, the right member of a rule can be represented as a graph whose name is the left member of the rule.

However, Unitex grammars are not exactly extended algebraic grammars, since they contain the notion of *transduction*. This notion, which is derived from the field of finite state automata, enables a grammar to produce some output. With an eye towards clarity, we will use the terms grammar or graph. When a grammar produces outputs, we will use the term *transducer*, as an extension of the definition of a transducer in the area of finite state automata.

5.2 Editing graphs

5.2.1 Import of Intex graphs

In order to be able to use Intex graphs in Unitex, they have to be converted to Unicode. The conversion procedure is the same as the one for texts (see section 2.2).

ATTENTION: A graph converted to Unicode that was used in Unitex cannot be used in Intex any longer.

In order to use it again in Intex, you have to convert it into ASCII and replace the first line:

```
#Unigraph
```

by the following line:

```
#FSGraph 4.0
```

5.2.2 Creating a graph

In order to create a graph, click on "New" in the "FSGraph" menu. You will then see the window coming up as in figure 5.2. The symbol in arrow form is the *initial state* of the graph. The round symbol with a square is the *final state* of the graph. The grammar only recognizes expressions that are described along the paths between initial and final states.

Figure 5.1: FSGraph menu

In order to create a box, click inside the window while pressing the Ctrl key. A blue rectangle will appear that symbolizes the empty box that was created (see figure 5.3). After creating the box, it is automatically selected.

You see the contents of that box in the text field at the top of the window. The newly created box contains the `<E>` symbol that represents the empty word epsilon. Replace this symbol by the text `I+you+he+she+it+we+they` and press the Enter key. You see that the box now contains seven lines (see figure 5.4). The `+` character serves as a separator. The box is displayed in the form of red text lines since it is not connected to another one at the moment. We often use this type of boxes to insert comments into a graph.

To connect a box to another one, first click on the source box, then click on the target box. If there already exists a transition between two boxes, it is deleted. It is also possible to use this operation by clicking first on the target box and then on the source box while pressing Shift. In our example, after connecting the box to the initial and final states of the graph, we get a graph as in figure 5.5:

Figure 5.2: Empty graph

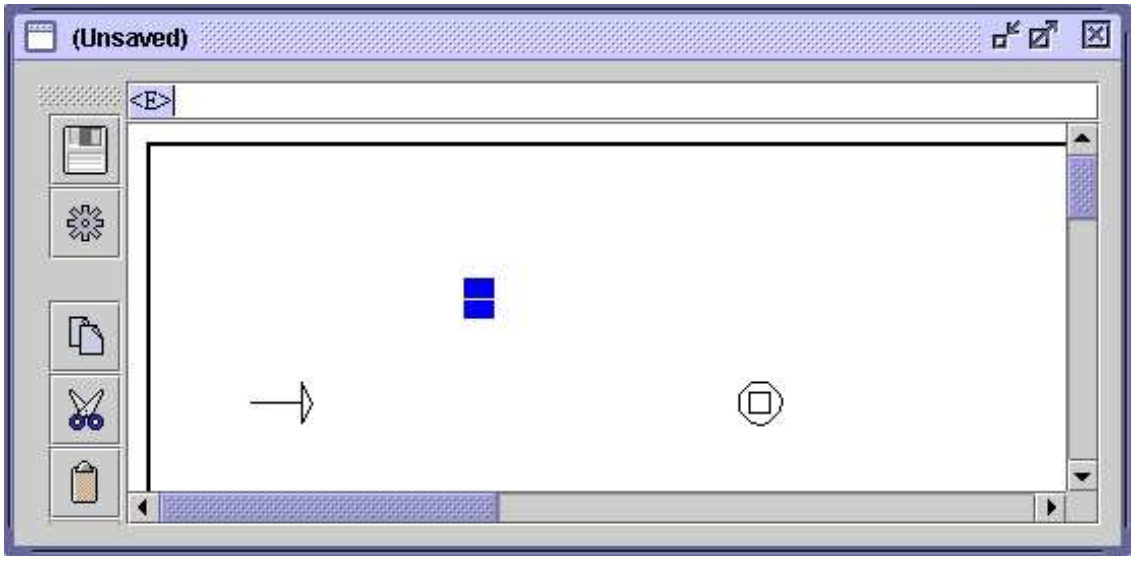


Figure 5.3: Creating a box

NOTE: If you double-click a box, you connect this box to itself (see figure 5.6). To undo this double-click on the same box a second time, or use the "Undo" button.

Click on "Save as..." in the "FSGraph" menu to save the graph. By default, Unitex proposes to save the graph in the sub-directory `Graphs` in your personal folder. You can see if the graph was modified after the last saving by checking if the title contains the text

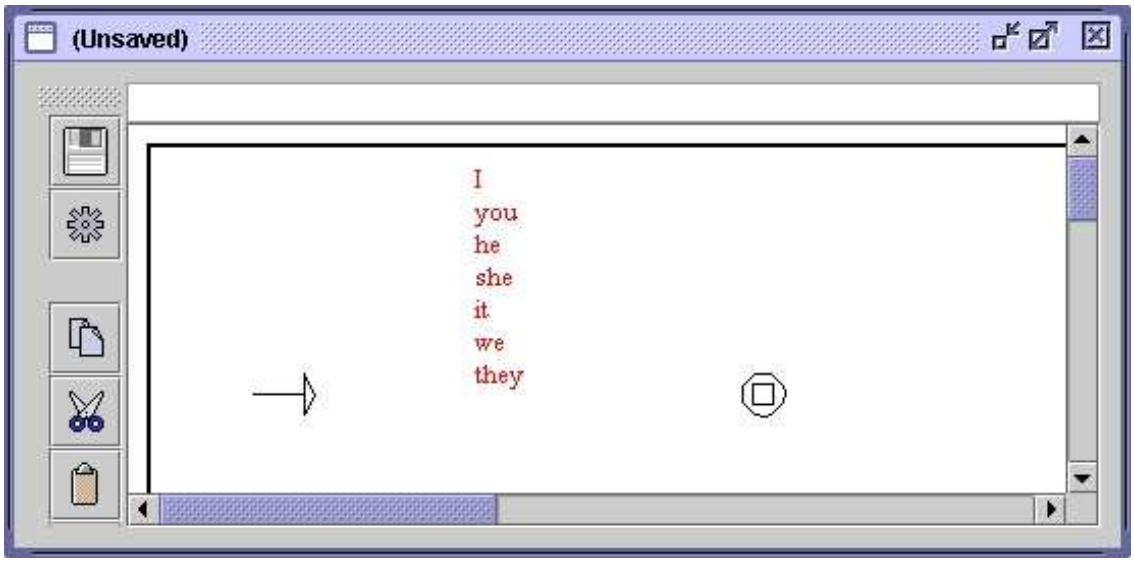


Figure 5.4: Box containing `I+you+he+she+it+we+they`

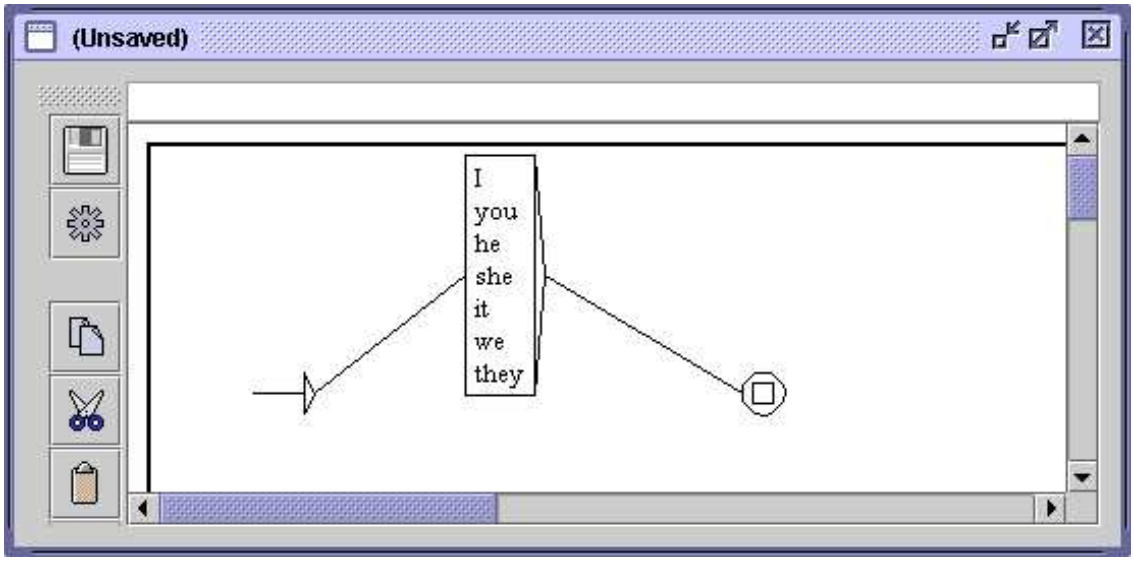


Figure 5.5: Graph that recognizes English pronouns

`(Unsaved).`

5.2.3 Sub-Graphs

In order to call a sub-graph, its name is inserted into a box and preceded by the `:` character. If you enter the text:

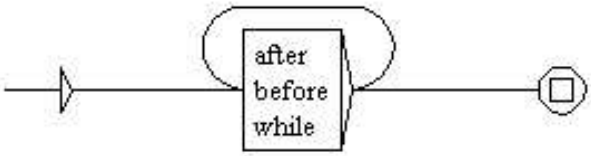


Figure 5.6: Box connected to itself

```
alpha+:beta+gamma+:E:\greek\delta.grf
```

into a box, you get a box similar to the one in figure 5.7:

Figure 5.7: Graph that calls sub-graphs `beta` and `delta`

You can indicate the complete name of the graph (`E:\greek\delta.grf`) or simply the base name without the path (`beta`); in this case, the the sub-graph is expected to be in the same directory as the graph that references it. References to absolute path names should as a rule be avoided, since such calls are not portable. If you use such an absolute path name, the graph compiler will emit a warning (see figure 5.8).

For portability you should not use `\` or `/` as separator in graph path names. Use instead `:` which is understood as a system-independent separator. In figure 5.8 `\` and `/` are internally converted by the graph compiler to `:` (`E::greek:delta.grf`).

Graph repository

When you need to call a grammar X inside a grammar Y, a simple method is to copy all the graphs of X into the directory that contains the graphs of Y. This method raises two problems:

- the number of graphs in the directory grows quickly;
- two graphs cannot share the same name.

To avoid that, you can store the grammar X in a special directory, called the *graph repository*. This directory is a kind of library where you can store graphs, and then call them using `::` instead of `:`. To use this mechanism, you first need to set the path to the graph repository.

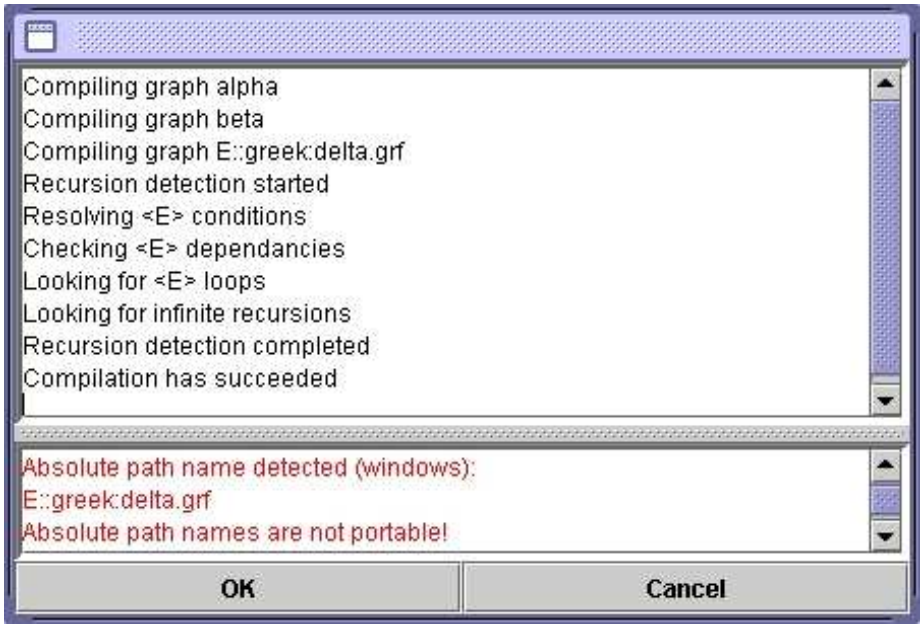


Figure 5.8: Avertissement pour un nom de graphe non portable

Go into the "Info>Preferences...>Directories" menu, and select your directory in the "Graph repository" frame (see Figure 5.9). There is one graph repository per language, so feel free to share or not the same directory for all the languages you work with.

Let us assume that we have a repository tree as on Figure 5.10. If we want to call the graph named `DET` that is located in sub-directory `Johnson`, we must use the call `::Det:Johnson:DET` (see Figure 5.11 [1]).

TRICK: If you want to avoid long path names like `::Det:Johnson:DET`, you can create a graph named `DET` and put it the repository root (here `D:\repository\DET.grf`). In this graph, just put a call to `::Det:Johnson:DET`. Then, you can just call `::DET` in your own graphs. This has two advantages: 1) you do not have long path names; 2) you can modify the graphs in your repository with no constrainst on your own graphs, because the only graph that will have to be modified is the one located at the repository root.

Calls to sub-graphs are represented in the boxes by grey lines, or brown lines in the case of graphs located in the repository. On Windows you can open a sub-graph by clicking on the grey line while pressing the Alt key. On Linux, the combination <Alt+Click> is intercepted by the system.[2] In order to open a sub-graph, click on its name by pressing the left and the right mouse button simultaneously.

[1] To avoid confusion, graph calls that refer to the repository are displayed in brown instead of grey.
[2] If you are working on KDE, you can deactivate <Alt+Click> in kcontrol.

Figure 5.9: Setting the path to the graph repository

Figure 5.10: Graph repository example

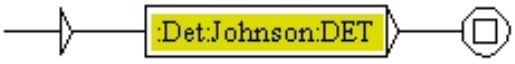


Figure 5.11: Call to a graph located in the repository

5.2.4 Manipulating boxes

You can select several boxes using the mouse. In order to do so, click and drag the mouse without releasing the button. When you release the button, all boxes touched by the selection rectangle will be selected and are displayed in white on blue ground:

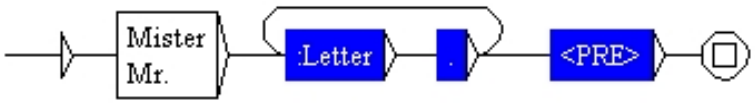


Figure 5.12: Selecting multiple boxes

When the boxes are selected, you can move them by clicking and dragging the cursor without releasing the button. In order to cancel the selection, click on an empty area of the graph. If you click on a box, all boxes of the selection will be connected to it.

You can perform a copy-paste using several boxes. Select them and press <Ctrl+C> or click on "Copy" in the "Edit" menu. The selection is now in the Unitex clipboard. You can then paste this selection by pressing <Ctrl+V> or by selecting "Paste" in the "Edit" menu.

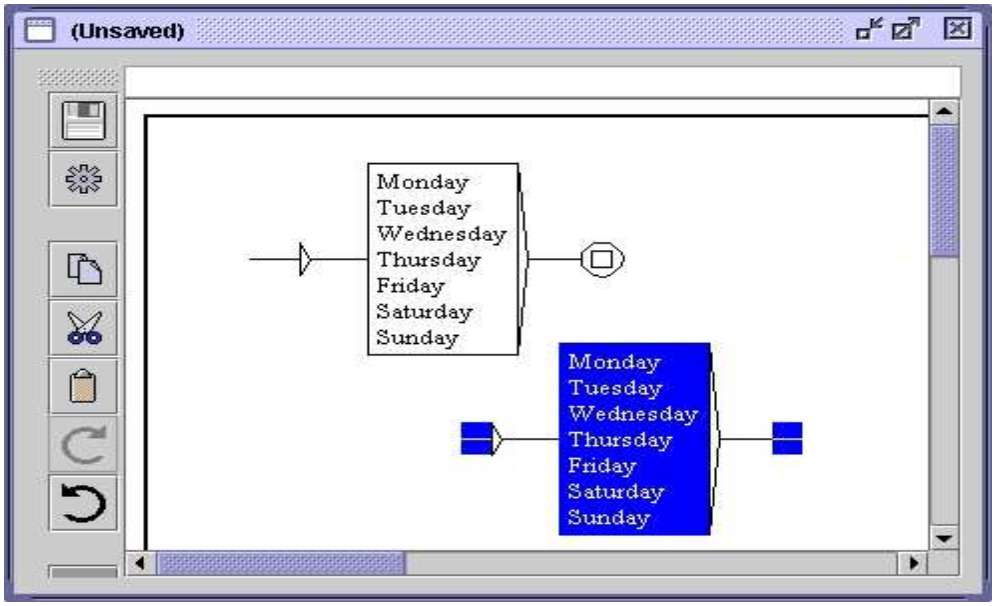


Figure 5.13: Copy-Paste of a multiple selection

NOTE: You can paste a multiple selection into a different graph than the one where you copied it from.

In order to delete boxes, select them and delete the text that they contain. Delete the text presented in the text field above the window and press the enter key. The initial and final states cannot be deleted.

5.2.5 Transducers

A transducer is a graph in which outputs can be associated with boxes. To insert aa output, use the special character `/`. All characters to the right of it will be part of the output. Thus, the text `one+two+three/number` results in a box like in figure 5.14:

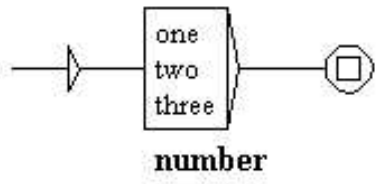


Figure 5.14: Example of a transducer

The output associated with a box is represented in bold text below it.

5.2.6 Using Variables

It is possible to select parts of a recognized text by a grammar using variables. To associate a variable `var1` with parts of a grammar, use the special symbols `$var1(` and `$var1)` to define the beginning and the end of the part to store. Create two boxes containing one `$var1(` and the second `$var1)`. These boxes must not contain anything but the variable name preceded by `$` and followed by a parenthesis. Then link these boxes to the zone of the grammar to store. In the graph in figure 5.15 you see a sequence of digits before `dollar` or `dollars`. This sequence will be stored in a variable named `var1`.

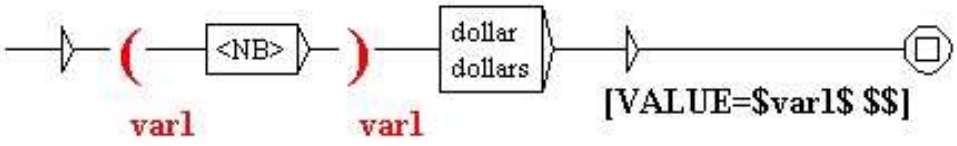


Figure 5.15: Using the variable `var1`

Variable names may contain latin letters (without accents), upper or lower case, numbers, or the _ (underscore) character. Unitex distinguishes between uppercase and lowercase characters.

When a variable is defined, you can use it in transducer outputs by surrounding its name with $. The grammar in figure 5.16 recognizes a date formed by a month and a year, and produces the same date as an output, but in the order year-month.

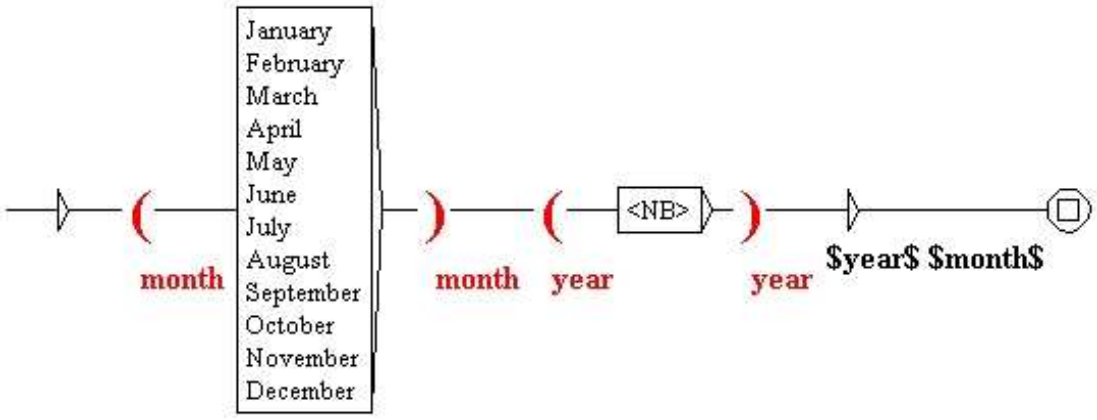


Figure 5.16: Inverting month and year in a date

If you want to use the character $ in the output of a box, you have to double it, as shown on figure 5.15.

5.2.7 Copying lists

It can be practical to perform a copy-paste operation on a list of words or expressions from a text editor to a box in a graph. In order to avoid having to copy every term manually, Unitex provides a means to copy lists. To use this, select the list in your text editor and copy it using <Ctrl+C> or the copy function integrated in your editor. Then create a box in your graph, and press <Ctrl+V> or use the "Paste" command in the "Edit" menu to paste it into the box. A window as in figure 5.17 opens:

Figure 5.17: Selecting a context for copying a list

This window allows you to define the left and right contexts that will automatically be used for each term of the list. By default, these contexts are empty. If you use the contexts `<` and `.V>` with the following list:

eat
sleep
drink
play
read

you will get the box in figure 5.18:

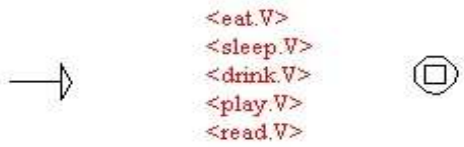


Figure 5.18: Box resulting from copying a list and applying contexts

5.2.8 Special Symbols

The Unitex graph editor interprets the following symbol in a special manner:

`" + : / < > # \`

Table 5.1 summarizes the meaning of these symbols for Unitex, as well as the places where these characters are recognized in the texts

Caracter	Meaning	Escape
"	quotation marks mark sequences that must not be interpreted by Unitex, and whose case must be taken verbatim	\"
+	+ separates different lines within the boxes	"+"
:	: introduces a call to a subgraph	":" or \:
/	/ indicates the start of a transduction within a box	\/
<	< indicates the start of a pattern or a meta	"<" or \<
>	> indicates the end of a pattern or a meta	">" or \>
#	# prohibits the presence of a space	"#"
\	\ escapes most of the special characters	\\

Table 5.1: Encoding of special characters in the graph editor

5.2.9 Toolbar Commands

The toolbar on the left of the graphs contains short cuts for certain commands and allows you to manipulate boxes of a graph by using some "utilities". This toolbar may be moved by clicking on the "rough" zone. It may also be dissociated from the graph and appear in an separate window (see figure 5.19). In this case, closing this window puts the toolbar back at its initial position. Each graph has its own toolbar.

Figure 5.19: Toolbar

The first two icons are shortcuts for saving and compiling the graph. The following five correspond to the Copy, Cut, Paste, Redo and Undo operations. The last icon showing a key is a shortcut to open the window with the graph display options.

The other six icons correspond to edit commands for boxes. The first one, a white arrow, corresponds to the boxes' normal edit mode. The 5 others correspond to specific utilities. In order to use a utility, click on the corresponding icon: The mouse cursor changes its form and mouse clicks are then interpreted in a particular fashion. What follows is a description of these utiltities, from left to right:

- creating boxes: creates a box at the empty place where the mouse was clicked;
- deleting boxes: deletes the box that you click on;
- connect boxes to another box: using this utility you select one or more boxes and connect it or them to another one. In contrast to the normal mode, the connections are inserted to the box where the mouse button was released on;
- connect boxes to another box in the opposite direction: this utility performs the same operation as the one described above, but connects the boxes to the one clicked on in opposite direction;
- open a sub-graph: opens a sub-graph when you click on a grey line within a box.

5.3 Display options

5.3.1 Sorting the lines of a box

You can sort the contents of a box by selecting it and clicking on "Sort Node Label" in the "Tools" submenu of the "FSGraph" menu. This sort operation doesn't use the `SortTxt` program. It uses a basic sort mechanism that sorts the lines of the box according to the order of the characters in the Unicode encoding.

5.3.2 Zoom

The "Zoom" submenu allows you to choose the zoom scale that is applied to display the graph.

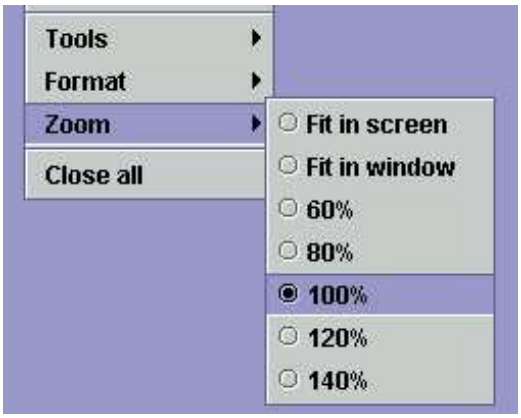


Figure 5.20: Zoom sub-menu

The option "Fit in screen" stretches or shrinks the graph in order to fit it into the screen. The option "Fit in window" adjusts the graph so that it is displayed completely in the window.

5.3.3 Antialiasing

Antialiasing is a shading effect that avoids pixelisation effects. You can activate this effect by clicking on "Antialiasing..." in the "Format" sub-menu. Figure 5.21 shows one graph displayed normally (the graph on top) and with antialiasing (the graph at the bottom).

This effect slows Unitex down. We recommend not to use it if your machine is not powerful enough.

5.3.4 Box alignment

In order to get nice-looking graphs, it is useful to align the boxes, both horizontally and vertically. To do this, select the boxes to align and click on "Alignment..." in the "Format" sub menu of the "FSGraph" menu or press <Ctrl+M>. You will then see the window in figure 5.22.

The possibilities for horizontal alignment are:

- Top: The boxes are aligned with the top-most box;
- Center: The boxes are centered with the same axis;
- Bottom: The boxes are aligned with the bottom-most box.

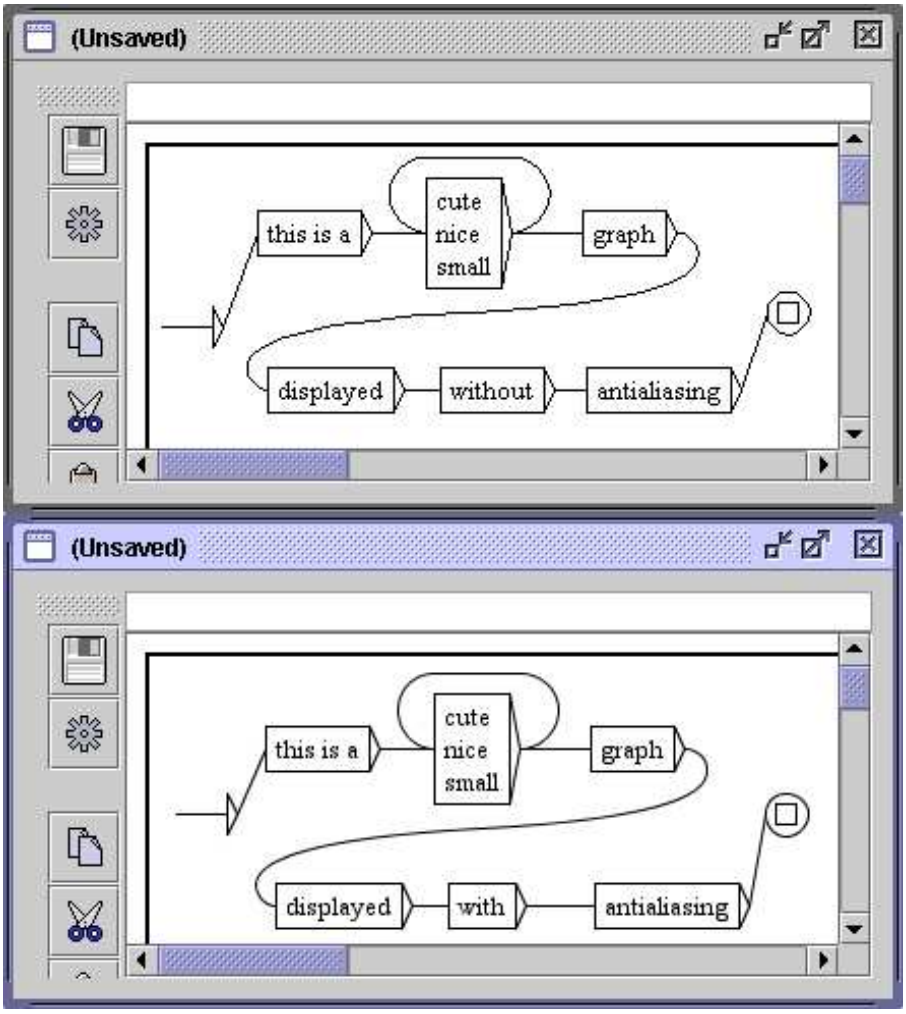


Figure 5.21: Antialiasing example

The possibilities for vertical alignment are:

- Left: The boxes are aligned with the left-most box;
- Center: The boxes are centered with the same axis;
- Right: The boxes are aligned with the right-most box.

Figure 5.23 shows an example of alignment. The group of boxes to the right is a copy of the ones to the left that was aligned vertically to the left.

The option "Use Grid" in the alignment window shows a grid as the background of the graph. This allows you to approximately align the boxes.

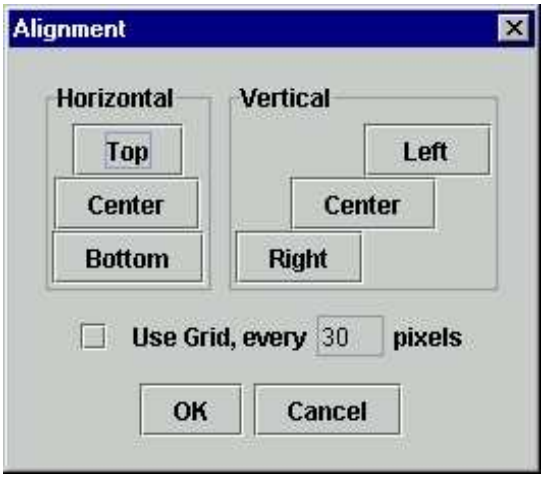


Figure 5.22: Alignment window

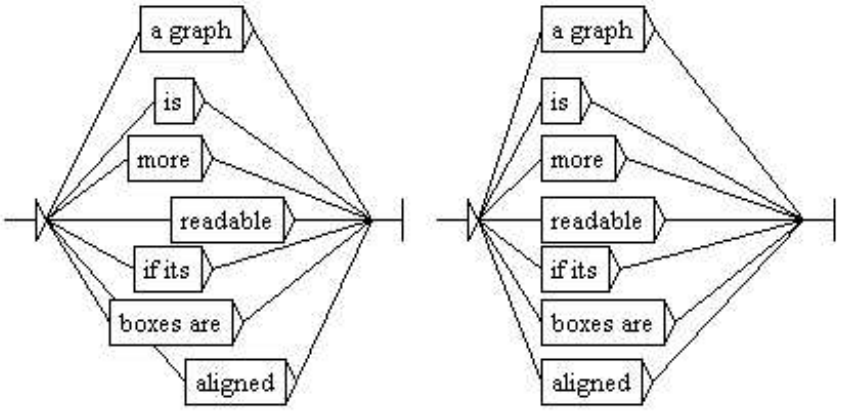


Figure 5.23: Example of aligning vertically to the left

5.3.5 Display options, fonts and colors

You can configure the display style of a graph by pressing <Ctrl+R> or by clicking on "Presentation..." in the "Format" sub-menu of the "FSGraph" menu, which opens the window as in figure 5.25.

The font parameters are:

- Input: Font used within the boxes and in the text area where the contents of the boxes is edited;
- Output: font used for the attached transducer outputs.

The color parameters are:

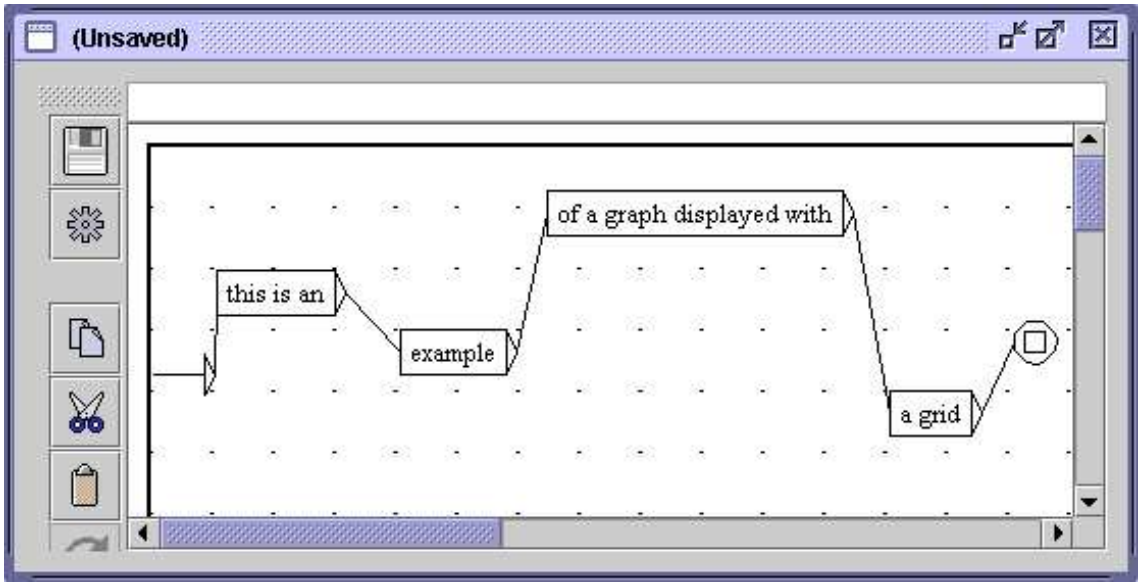


Figure 5.24: Example of using the grid

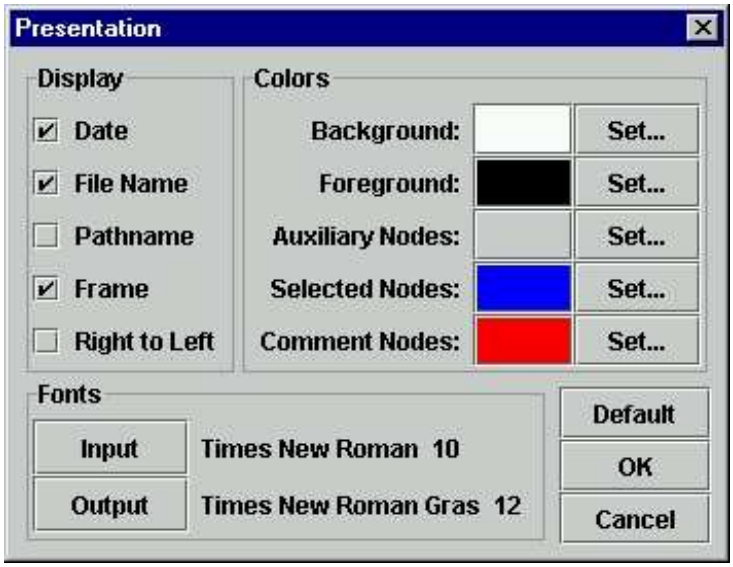


Figure 5.25: Configuring the display options of a graph

- Background: the background color;
- Foreground: the color used for the text and for the box display;
- Auxiliary Nodes: the color used for calls to sub-graphs;
- Selected Nodes: the color used for selected boxes;

- Comment Nodes: the color used for boxes that are not connected to others.

The other parameters are:

- Date: display of the current date in the lower left corner of the graph;
- File Name: display of the graph name in the lower left corner of the graph;
- Pathname: display of the graph name along with its complete path in the lower left corner of the graph. This option only has an effect if the option "File Name" is selected;
- Frame: draw a frame around the graph;
- Right to Left: invert the reading direction of the graph (see an example in figure 5.26).

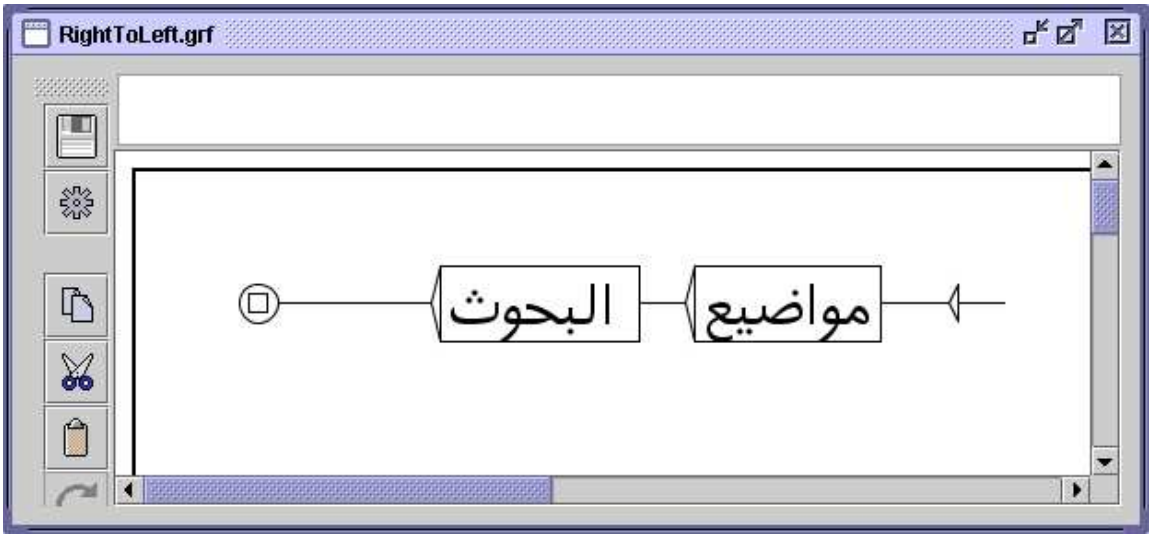


Figure 5.26: Graph with reading direction set to right to left

You can reset the parameters to the default ones by clicking on "Default". If you click on "OK", only the current graph will be modified. In order to modify the preferences for a language as a default, click on "Preferences..." in the "Info" menu and choose the tab "Graph Representation".

The preferences configuration window has an extra option concerning antialiasing (see figure 5.27). This option activates antialiasing by default for all graphs in the current language. It is advised to not activate this option if your machine is not very fast. You can also change the position of the icon bar.

NOTE: the "Right to Left" option is not present on the general graph configuration frame. The orientation of graphs is set per default for the current language, as defined in the "Text Presentation" tab (see Figure 4.7, page 62).)

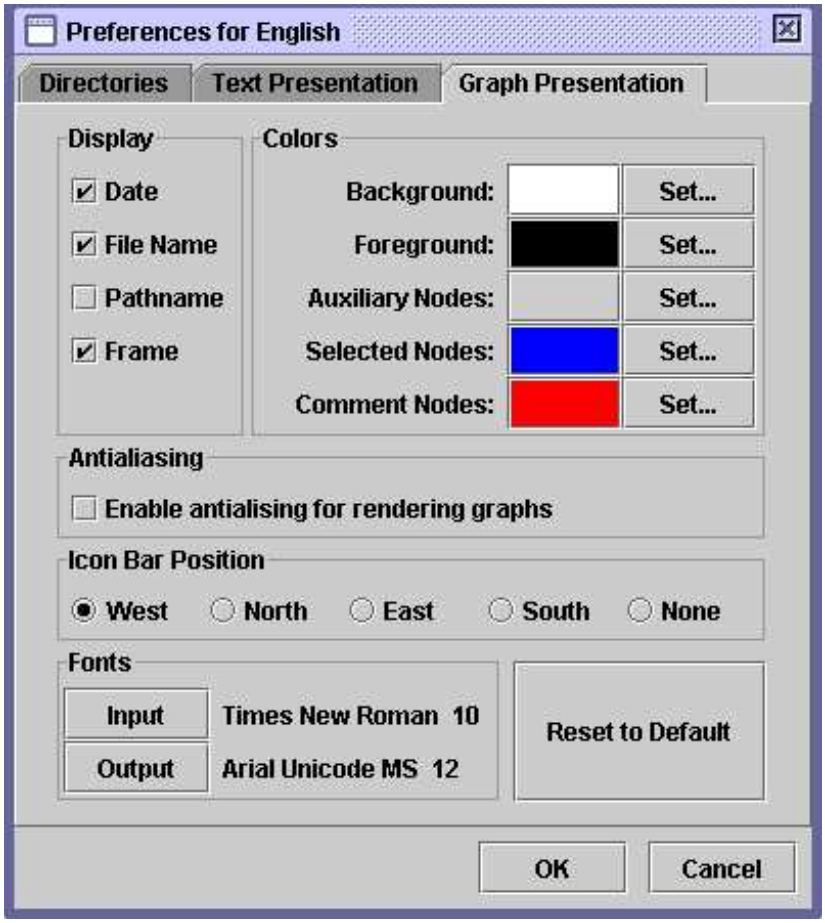


Figure 5.27: Default preferences configuration

5.4 Exporting graphs

5.4.1 Inserting a graph into a document

In order to include a graph into a document, you have to convert it to an image. To do this, save your graph as a PNG image. Click on "Save as..." in the "FSGraph" menu, and select the PNG file format. You will get an image ready to be inserted into a document, or to be edited with an image editor. You should activate antialiasing for the graph that interests you (this is not obligatory but results in a better image quality).

Another solution consists of making a screenshot:

On Windows:

Press "Print Screen" on your keyboard. This key should be next to the F12 key. Start the `Paint` program in the Windows "Utilities" menu. Press <Ctrl+V>. `Paint` will tell you that

the image in the clipboard is too large and asks if you want to enlarge the image. Click on "Yes". You can now edit the screen image. Select the area that interests you. To do so, switch to the select mode by clicking on the dashed rectangle symbol in the upper left corner of the window. You can now select the area of the image using the mouse. When you have selected the zone, press <Ctrl+C>. Your selection is now in the clipboard, you can now just go to your document and press <Ctrl+V> to paste your image.

On Linux:

Take a screen capture (for example using the program `xv`). Edit your image at once using a graphic editor (for example `TheGimp`), and paste your image in your document in the same way as in Windows.

5.4.2 Printing a Graph

You can print a graph by clicking on "Print..." in the "FSGraph" menu or by pressing <Ctrl+P>.

ATTENTION: You should make sure that the page orientation parameter (portrait or landscape) corresponds to the orientation of your graph.

You can specify the printing preferences by clicking on "Page Setup" in the "FSGraph" menu. You can also print all open graphs by clicking on "Print All...".

Chapter 6

Advanced use of graphs

6.1 Types of graphs

Unitex can work with several types of graphs that correspond to the following uses: automatic inflection of dictionaries, preprocessing of texts, normalization of text automata, dictionary graphs, search for patterns, disambiguation and automatic graph generation. These different types of graphs are not interpreted in the same way by Unitex. Certain operations, like transduction, are allowed for some types and forbidden for others. In addition, the special symbols are not the same depending on the type of the graph. This section presents each type of graph and shows their peculiarities.

6.1.1 Inflection transducers

An inflection transducer describes the morphological variation that is associated with a word class by assigning inflectional codes to each variant. The paths of such a transducer describe the modifications that have to be applied to the canonical forms and the corresponding outputs contain the inflectional information that will be produced.

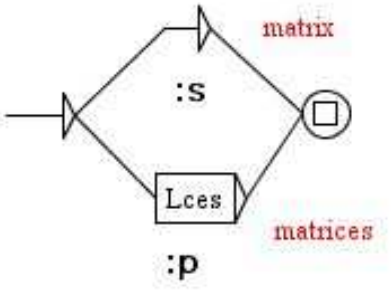


Figure 6.1: Example of an inflectional grammar

The paths may contain operators and letters. The possible operators are represented by the characters `L`, `R` and `C`. All letters that are not operators are characters. The only

allowed special symbol is the empty word <E>. It is not possible to refer to information in dictionaries in an inflection transducer. It is also impossible to reference subgraphs.

Transducer outputs are concatenated in order to produce a string of characters. This string is then appended to the line of the produced dictionary. Outputs with variables do not make sense in an inflection transducer.

The contents of an inflection transducer are manipulated without a change of case: lowercase letters stay lowercase, the same for uppercase letters. Besides, the connection of two boxes is exactly equivalent to the concatenation of their contents together with the concatenation of their outputs. (cf. figure 6.2).

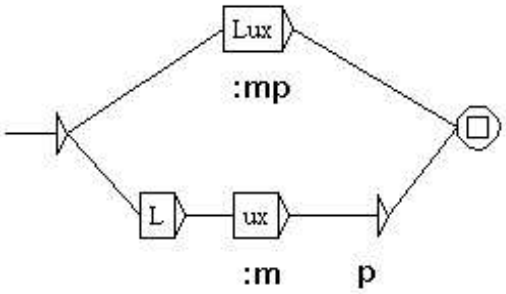


Figure 6.2: Two equivalent paths in an inflection grammar

Inflection transducers have to be compiled before being used by the inflection program.

For more details, see section 3.4.

6.1.2 Preprocessing graphs

Preprocessing graphs are meant to be applied to texts before they are tokenized into lexical units. These graphs can be used for inserting or replacing sequences in the texts. The two customary uses of these graphs are normalization of non-ambiguous forms and sentence boundary recognition.

The interpretation of these graphs in Unitex is very close to that of syntactic graphs used by the search for patterns. The differences are the following:

- you can use the special symbol <^> that recognizes a newline;
- it is impossible to refer to information in dictionaries;
- it is impossible to use morphological filters;
- it is impossible to use contexts.

The figures 2.9 (page 23) and 2.10 (page 25) show examples of preprocessing graphs.

6.1.3 Graphs for normalizing the text automaton

The graphs for normalization of the text automaton allow you to normalize ambiguous forms. They can describe several labels for the same form. These labels are then inserted into the text automaton thus making the ambiguity explicit. Figure 6.3 shows an extract of the normalization graph used by default for French.

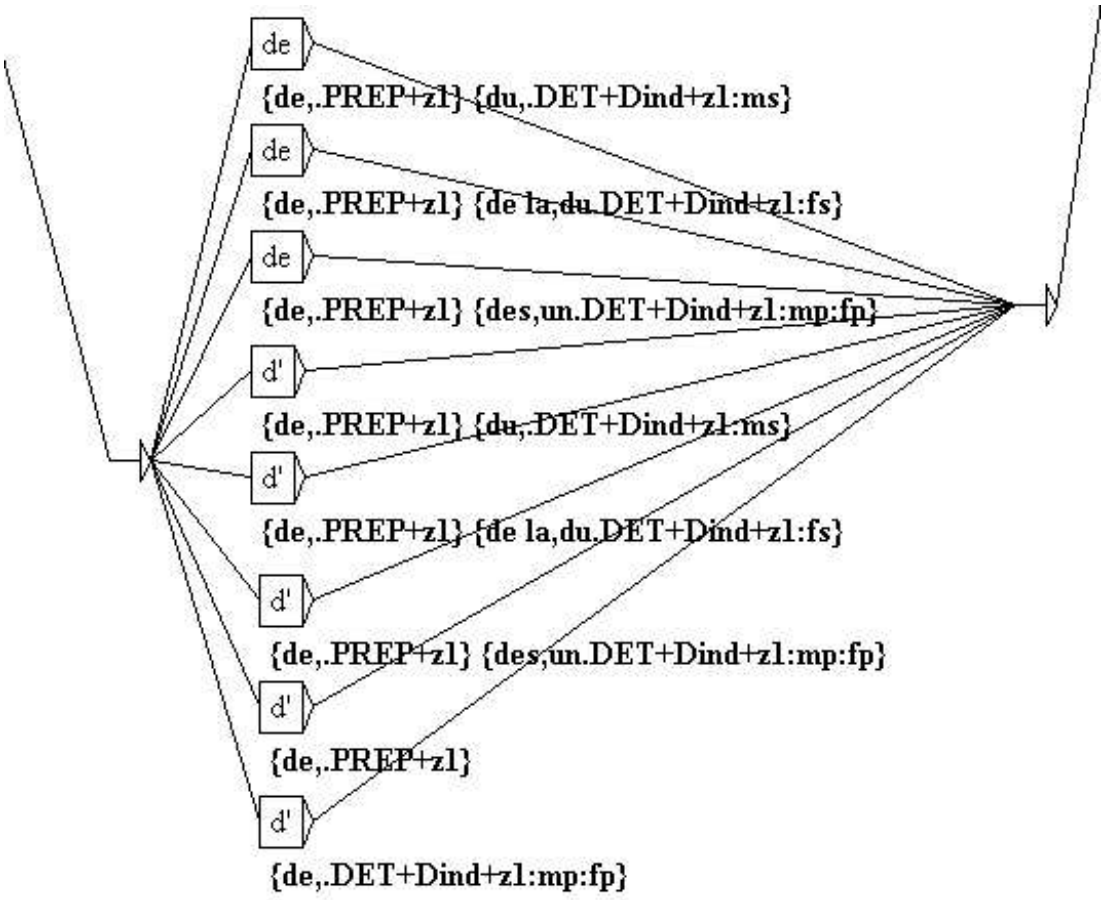


Figure 6.3: Extract of the normalization graph used for French

The paths describe the forms that have to be normalized. Lower case and upper case variants are taken into account according to the following principle: uppercase letters in the graph only recognize uppercase letters in the text automaton; lowercase letters can recognize lowercase and uppercase letters.

The transducer outputs represent the sequences of labels that will be inserted into the text automaton. These labels can be dictionary entries or strings of characters. The labels that represent entries of the dictionary have to respect the format for entries of a DELAF and are enclosed by the symbols `{` and `}`. Outputs with variables do not make sense in this kind of graph. You can not use morphological filters or contexts.

It is possible to reference subgraphs. It is not possible to reference information in dictionaries in order to describe the forms to normalize. The only special symbol that is recognized in this type of graph is the empty word <E>. The graphs for normalizing ambiguous forms need to be compiled before using them.

6.1.4 Syntactic graphs

Syntactic graphs, often called local grammars, allow you to describe syntactic patterns that can then be searched in the texts. Of all kinds of graphs these have the greatest expressive power because they allow you to refer to information in dictionaries.

Lower case/upper case variants may be used according to the principle described above. It is still possible to enforce respect of case by enclosing an expression in quotes. The use of quotes also allows you to enforce the respect of spaces. In fact, Unitex by default assumes that a space is possible between two boxes. In order to enforce the presence of a space you have to enclose it in quotes. For prohibiting the presence of a space you have to use the special symbol #.

The syntactic graphs can reference subgraphs (cf. section 5.2.3). They also have outputs including outputs with variables. The produced sequences are interpreted as strings of characters that will be inserted in the concordances or in the text if you want to modify it (cf. section 6.7.3).

Syntactic graphs can use contexts (see section 6.3).

The special symbols that are supported by the syntactic graphs are the same as those that are usable in regular expressions (cf. section 4.3.1).

It is not obligatory to compile syntactic graphs before using them for pattern searching. If a graph is not compiled the system will compile it automatically.

6.1.5 ELAG grammars

ELAG grammars for disambiguation between lexical symbols in text automata are described in section 7.3.1, page 118.

6.1.6 Parameterized graphs

The parameterized graphs are meta-graphs that allow you to generate a family of graphs using a lexicon-grammar table. It is possible to construct parameterized graphs for all possible kinds of graphs. The construction and use of parameterized graphs will be explained in chapter 8.

6.2 Compilation of a grammar

6.2.1 Compilation of a graph

Compilation is the operation that converts the `.grf` format to a format that can be manipulated more easily by the Unitex programs. In order to compile a graph you open it and then

click on "Compile FST2" in the submenu "Tools" of the menu "FSGraph". Unitex then starts the program `Grf2Fst2`. You can keep track of its execution in a window (cf. figure 6.4).

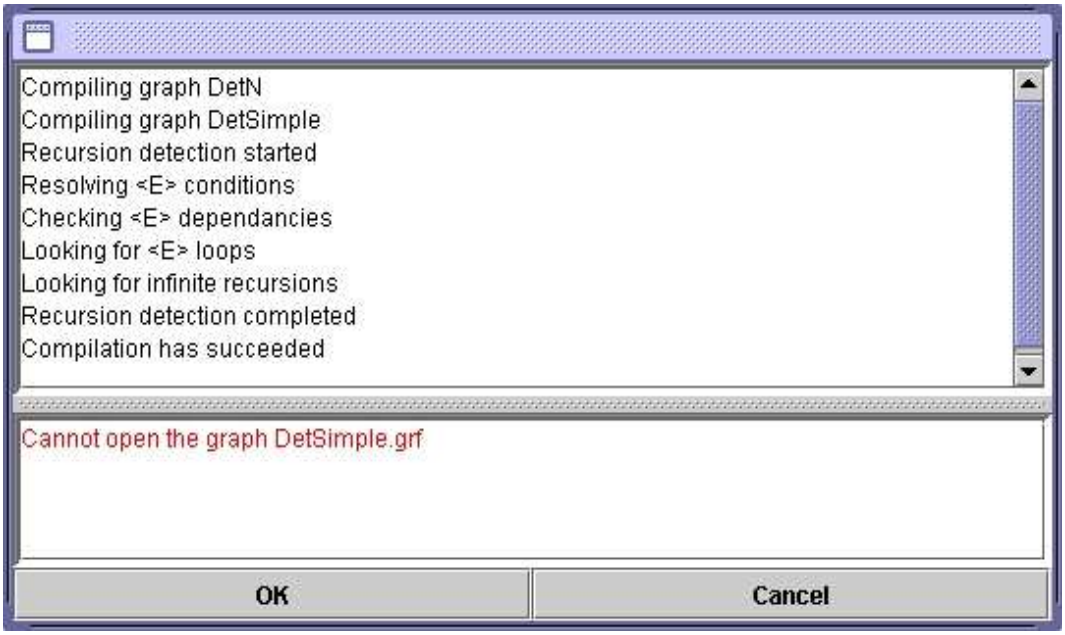


Figure 6.4: Compilation window

If the graph references subgraphs, those are automatically compiled. The result is a `.fst2` file that contains all the graphs that make up a grammar. The grammar is then ready to be used by the different Unitex programs.

6.2.2 Approximation with a finite state transducer

The FST2 format conserves the architecture in subgraphs of the grammars, which is what makes them different from strict finite state transducers. The program `Flatten` allows you to transform a grammar FST2 into a finite state transducer whenever this is possible, and to construct an approximation if not. This function thus permits to obtain objects that are easier to manipulate and to which all classical algorithms on automata can be applied.

In order to compile and thus transform a grammar select the command "Compile & Flatten FST2" in the submenu "Tools" of the menu "FSGraph". The window of figure 6.5 allows you to configure the operation of approximation.

The box "Flattening depth" lets you specify the level of embedding of subgraphs. This value represents the maximum depth up to which the calling of subgraphs will be replaced by the subgraphs themselves.

The "Expected result grammar format" box allows you to determine the behavior of the program beyond the selected limit. If you select the option "Finite State Transducer", the

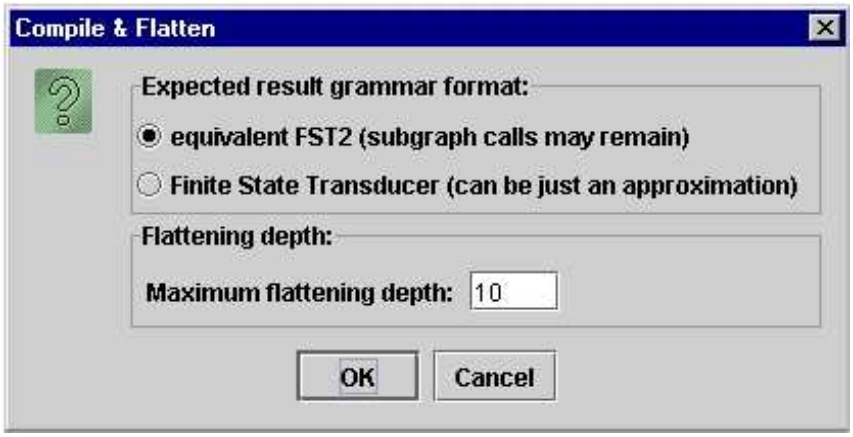


Figure 6.5: Configuration of approximation of a grammar

calls to subgraphs will be replaced by <E> beyond the maximum depth. This option guarantees that we obtain a finite state transducer, however possibly not equivalent to the original grammar. On the contrary, the "equivalent FST2" option indicates that the program should allow for subgraph calls beyond the limited depth. This option guarantees the strict equivalence of the result with the original grammar but does not necessarily produce a finite state transducer. This option can be used for optimizing certain grammars.

A message indicates at the end of the approximation process if the result is a finite state transducer or an FST2 grammar and in the case of a transducer if it is equivalent to the original grammar (cf. figure 6.6).

6.2.3 Constraints on grammars

With the exception of inflection grammars, a grammar can never have an empty path. This means that the paths of a principal grammar must not recognize the empty word but this does not prevent a subgraph of that grammar from recognizing epsilon.

It is not possible to associate a transducer output with a call to a subgraph. Such outputs are ignored by Unitex. It is therefore necessary to use an empty box that is situated to the left of the call to the subgraph in order to specify the output (cf. figure 6.7).

The grammars must not contain void loops because the Unitex programs cannot terminate the exploration of such a grammar. A void loop is a configuration that causes the `Locate` program to enter an infinite loop. Void loops can originate from transitions that are labeled by the empty word or from recursive calls to subgraphs.

Void loops due to transitions with the empty word can have two origins of which the first is illustrated by the figure 6.8.

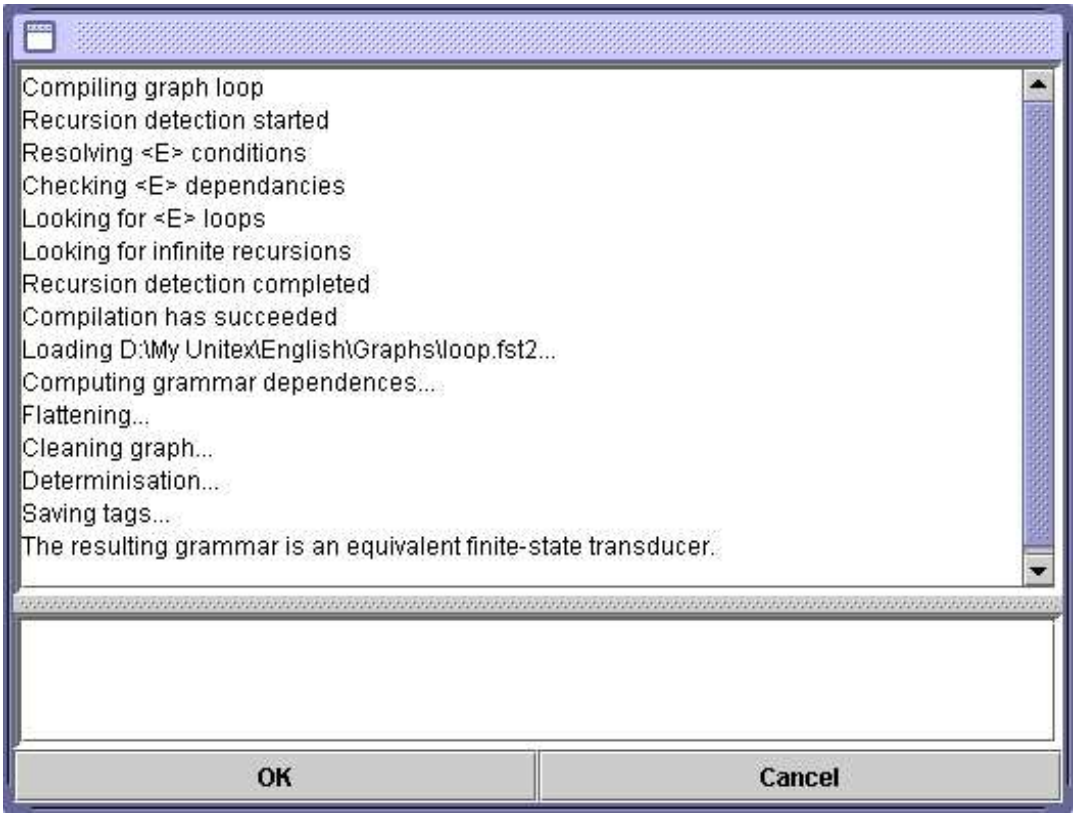


Figure 6.6: Resultat of the approximation of a grammar

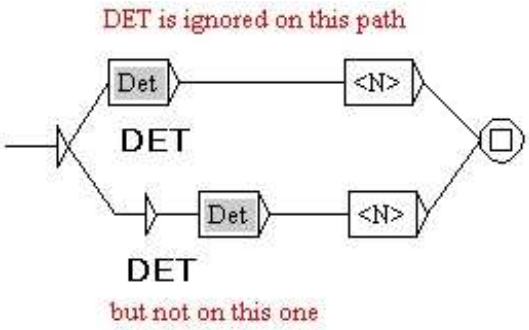


Figure 6.7: How to associate an output with a call to a subgraph

This type of loops is due to the fact that a transition with the empty word cannot be eliminated automatically by Unitex because it is associated with an output. Thus, the transition with the empty word of figure 6.8 will not be suppressed and will cause a void loop.

The second category of loop by epsilon concerns the call to subgraphs that can recognize

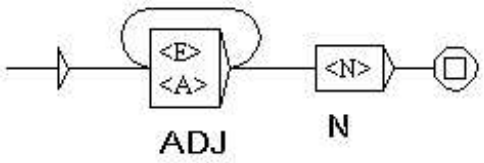


Figure 6.8: Void loop due to a transition by the empty word with a transduction

the empty word. This case is illustrated in figure 6.9: if the subgraph `Adj` recognizes epsilon, there is a void loop that Unitex cannot detect.

Figure 6.9: Void loop due to a call to a subgraph that recognizes epsilon

The third possibility of void loops is related to recursive calls to subgraphs. Look at the graphs `Det` and `DetCompose` in figure 6.10. Each of these graphs can call the other *without reading any text*. The fact that none of these two graphs has labels between the initial state and the call to the subgraph is crucial. In fact, if there were at least one label different from epsilon between the beginning of the graph `Det` and the call to `DetCompose`, this would mean that the Unitex programs exploring the graph `Det` would have to read the pattern described by that label in the text before calling `DetCompose` recursively. In this case the programs would loop infinitely only if they recognized the pattern an infinite number of times in the text, which is impossible.

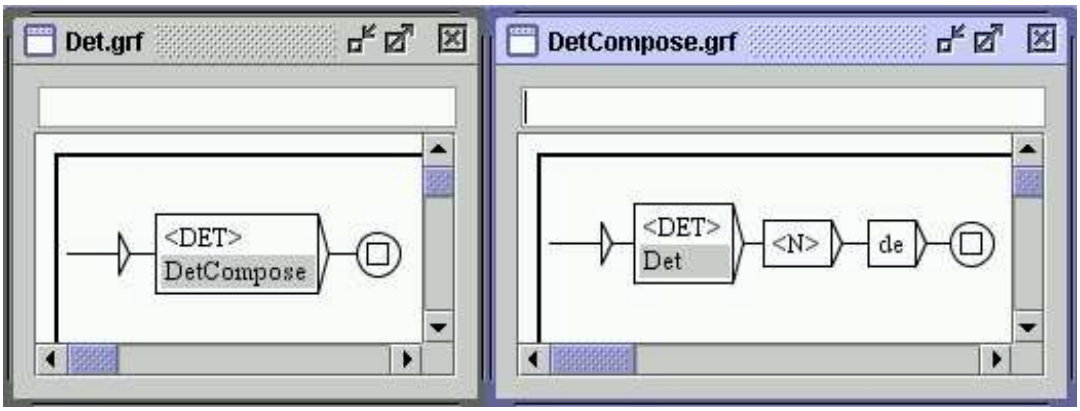


Figure 6.10: Void loop caused by two graphs calling each other

6.2.4 Error detection

In order to keep the programs from blocking or crashing, Unitex automatically detects errors during graph compilation. The graph compiler verifies that the principal graph does not recognize the empty word and searches for all possible forms of infinite loops. When an error is encountered an error message is displayed in the compilation window. Figure 6.11 shows the message that appears if one tries to compile the graph `Det` of figure 6.10.

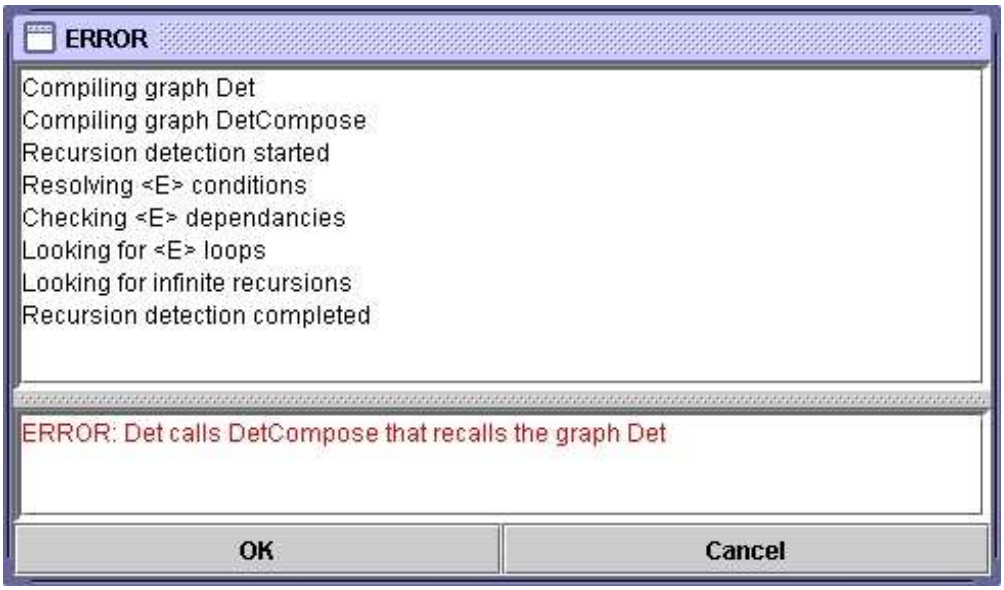


Figure 6.11: Error message when trying to compile `Det`

If you have started a pattern search be selecting a graph of the format `.grf` and Unitex discovers an error, the operation is automatically interrupted.

6.3 Contexts

Unitex graphs as we described them up to there are equivalent to algebraic grammars. These are also known as context-free grammars, because if you want to match a sequence A, the context of A is irrelevant. Thus, you cannot use a contex-free graph for matching occurences of `president` not followed by `of the republic`.

However, you can draw graphs with positive or negative contexts. In that case, graphs are no more equivalent to algebraic grammars, but to context-sensitive grammars that do not have the same theoretical properties.

To define a context, you must bound a zone of the graph with boxes containing `$[` and `$]`, which indicate the start and the end of the context. These bounds appear in the graph as green square brackets. Both bounds of a context must be located in the same graph.

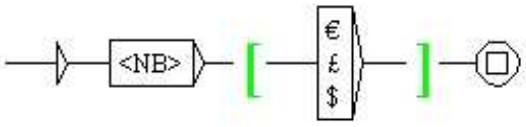


Figure 6.12: Using a context

Figure 6.12 shows a simple context. The graph matches numbers followed by a currency symbol, but this symbol will not appear in matched sequences, i.e. in the concordance.

Contexts are interpreted as follows. During the application of a grammar on a text, let us assume that a context start is found. Let *pos* be the current position in the text at this time. Now, the `Locate` program tries to match the expression described inside the context. If it fails, then there will be no match. If it matches the whole context (that is to say if `Locate` reaches the context end), then the program will rewind at the position `pos` and go on exploring the grammar after the context end.

You can also define negative contexts, using `$![` to indicate the context start. Figure 6.13 shows a graph that matches numbers that are not followed by `th`. The difference with postitive contexts is that when `Locate` tries to match the expression described inside the context, reaching the context stop will be considered as a failure, because it would have matched a forbidden sequence. At the opposite, if the context stop cannot be reached, then `Locate` will rewind at the position *pos* and go on exploring the grammar after the context end.

Figure 6.13: Using a negative context

Contexts can appear anywhere in the graph, including the beginning of the graph. Figure 6.14 shows a graph that matches an adjective in the context of something that is not a past participle. In other words, this graph matches adjectives that are not ambiguous with past participles.

This mechanism allows you to formulate complex patterns. For instance, the graph of figure 6.15 matches a sequence of two simple nouns that is not ambiguous with a compound word. In fact, the pattern `<CDIC><<^([^ ]+ [^ ]+)$>>` matches a compound

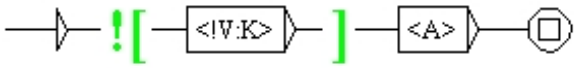


Figure 6.14: Matching an adjective that is not ambiguous with a past participle

word with exactly one space, and the pattern `<N><<^([^ ]+)$>>` matches a noun without space, that is to say a simple noun. Thus, in the sentence *Black cats should like the town hall*, this graph will match *Black cats*, but not *town hall*, which is a compound word.

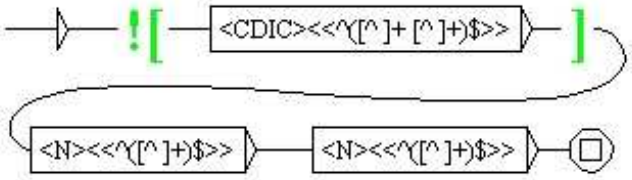


Figure 6.15: Advanced use of contexts

You can use nested contexts. For instance, the graph shown in figure 6.16 matches a number that is not followed by a point, except for a point followed by a number. Thus, in the sequence *5.0+7.=12*, this graph will match *5*, *0* and *12*.

Figure 6.16: Nested contexts

If a context contains boxes with transducer outputs, the outputs are ignored. However, it is possible to use a variable that was defined inside a context (cf. figure 6.17). If you apply this graph in MERGE mode to the text *the cat is white*, you will obtain:

```
the <pet name="cat" color="white"/> is white
```

6.4 Exploring grammar paths

It is possible to generate the paths recognized by a grammar, if they are in finite number, for example to verify that it correctly generates the expected forms. For that, open the main

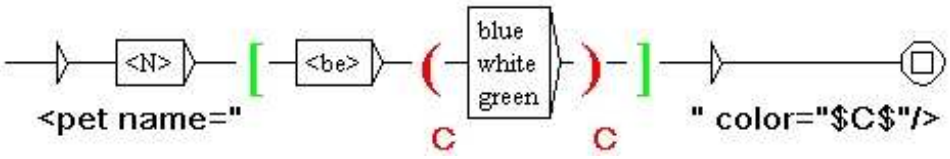


Figure 6.17: Variable defined inside a context

graph of your grammar, and ensure that the graph window is the active window (the active window has a blue title bar, while the inactive windows have a gray title bar). Now go to the "FSGraph" menu and then to the "Tools" menu, and click on "Explore Graph paths". The Window of figure 6.18 appears.

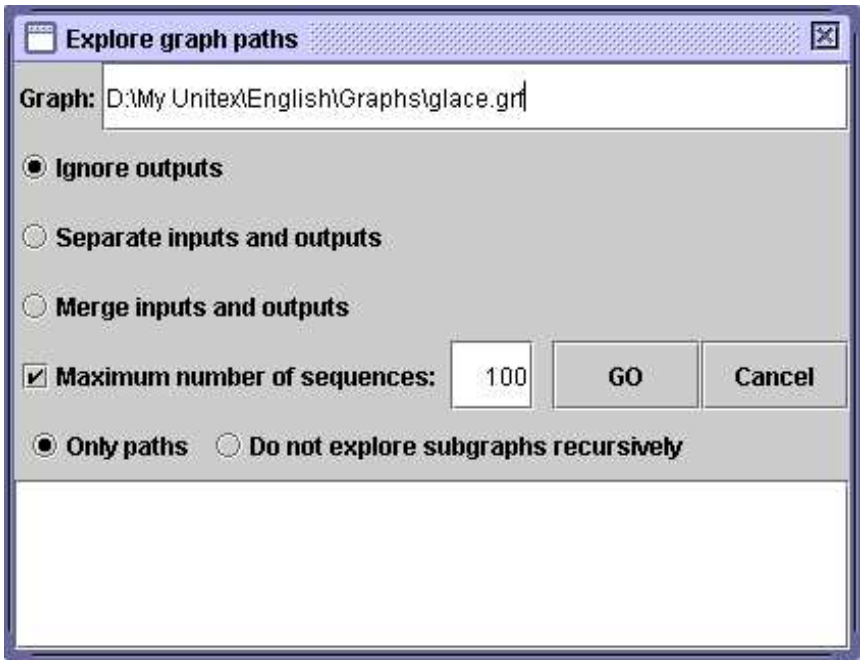


Figure 6.18: Exploring the paths of a grammar

The upper box contains the name of the main graph of the grammar to be explored. The following options are connected to the outputs of the grammar and to subgraph calls:

- "Ignore outputs": outputs are ignored;
- "Separate inputs and outputs": outputs are displayed after inputs (`a b c / A B C`);
- "merge inputs and outputs": Every output is emitted immediately after the input to which it corresponds (`a/A b/B c/C`).

- "Only paths": calls to subgraphs are explored recursively;
- "Do not explore subgraphs recursively": calls to subgraphs are printed but not explored recursively.

If the option "Maximum number of sequences" is activated, the specified number will be the maximum number of generated paths. If the option is not selected, all paths will be generated, if they are in finite number.

Here you see what is created for the graph in figure 6.19 with default settings (ignoring outputs, limit = 100 paths):

```
<NB> <boule> de glace à la pistache
<NB> <boule> de glace à la fraise
<NB> <boule> de glace à la vanille
<NB> <boule> de glace vanille
<NB> <boule> de glace fraise
<NB> <boule> de glace pistache
<NB> <boule> de pistache
<NB> <boule> de fraise
<NB> <boule> de vanille
glace à la pistache
glace à la fraise
glace à la vanille
glace vanille
glace fraise
glace pistache
```

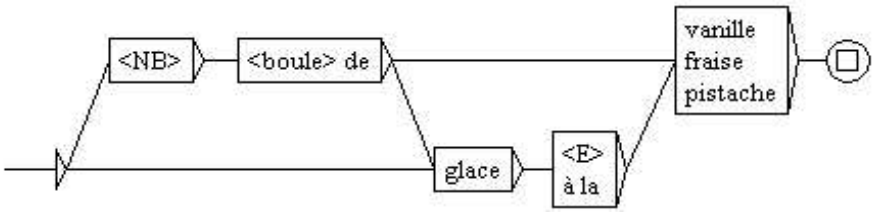


Figure 6.19: Sample graph

6.5 Graph collections

It can happen that one wants to apply several grammars located in the same directory. For that, it is possible to automatically build a grammar starting from a tree structure of files. Let us suppose for example that one has the following tree structure:

- *Dicos*:

- *Banque*:
  - `carte.grf`
- *Nourriture*:
  - `eau.grf`
  - `pain.grf`
- `truc.grf`

If one wants to gather all these grammars in only one, one can do it with the "Build Graph Collection" command in the "FSGraph Tools" sub-menu. One configures this operation by means of the window seen in figure 6.20.

Figure 6.20: Building a graph collection

In the "Source Directory" field, select the root directory which you want to explore (in our example, the directory *Dicos*). In the field "Resulting GRF grammar", enter the name of the produced grammar.

CAUTION: Do not place the output grammar in the tree structure which you want to explore, because in this case the program will try to read and to write simultaneously in this file, which will cause a crash.

When you click on "OK", the program will copy the graphs to the directory of the output grammar, and will create subgraphs corresponding to the various sub-directories, as one can see in figure 6.21, which shows the output graph generated for our example.

One can observe that one box contains the calls with subgraphs corresponding to sub-directories (here directories *Banque* and *Nourriture*), and that the other box calls all the graphs which were in the directory (here the graph `truc.grf`).

6.6 Rules for applying transducers

This section describes the rules for the application of transducers along with the operations of preprocessing and the search for patterns. The following does not apply to inflection graphs and normalization graphs for ambiguous forms.

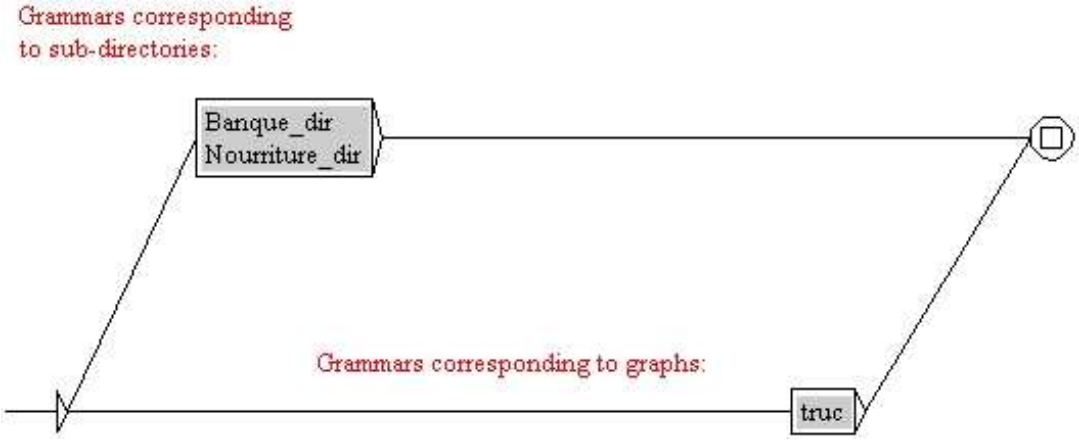


Figure 6.21: Main graph of a graph collection

6.6.1 Insertion to the left of the matched pattern

When a transducer is applied in REPLACE mode, the output replaces the sequences that have been read in the text. When a box in a transducer does not have an output, it is processed as if it had an <E> output. In MERGE mode, the output is inserted to the left of the recognized sequences. Look at the transducer in figure 6.22.

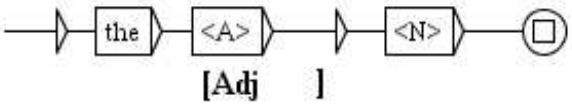


Figure 6.22: Example of a transducer

If this transducer is applied to the novel *Ivanhoe* by Sir Walter Scott in MERGE mode, the following concordance is obtained.

6.6.2 Application while advancing through the text

During the preprocessing operations, the text is modified as it is being read. In order to avoid the risk of infinite loops, it is necessary that the sequences that are produced by a transducer will not be re-analyzed by the same one. Therefore, whenever a sequence is inserted into the text, the application of the transducer is continued after that sequence. This rule only applies to preprocessing transducers, because during the application of syntactic graphs, the transductions do not modify the processed text but a concordance file which is distinct from the text.

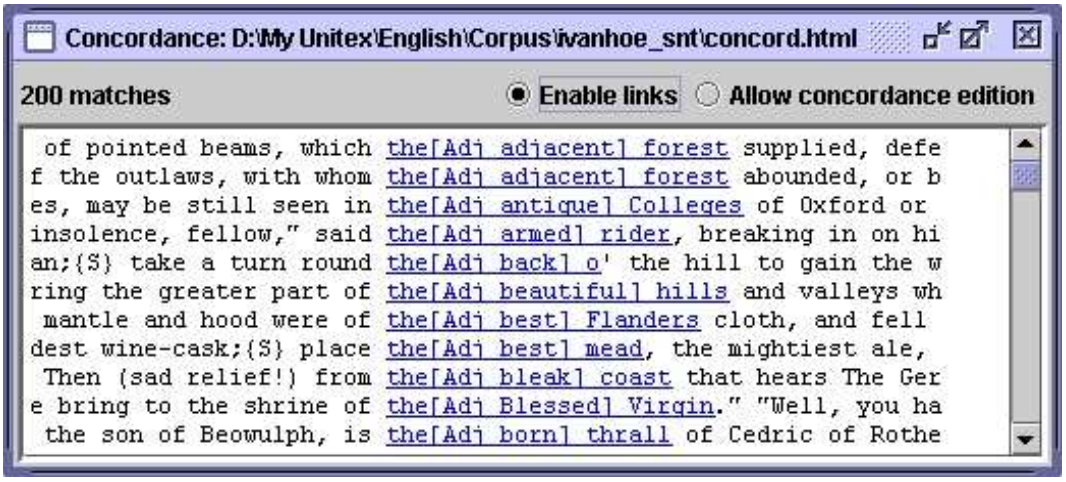


Figure 6.23: Concordance obtained in MERGE mode with the transducer of figure 6.22

6.6.3 Priority of the leftmost match

During the application of a local grammar, the overlapping occurrences are all indexed. During the construction of the concordance all these overlapping occurrences are presented (cf. figure 6.24).

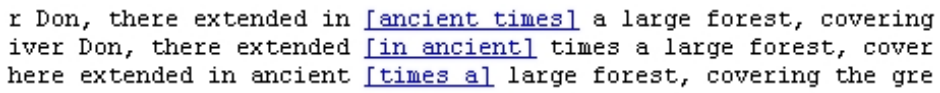


Figure 6.24: Occurrences are collected into concordance

On the other hand, if you modify a text instead of constructing a concordance, it is necessary to choose among these occurrences the one that will be taken into account. Unitex applies the following prioritisation rule for that purpose: the leftmost sequence is used.

If this rule is applied to the three occurrrences of the preceding concordance, the occurrence `[in ancient]` overlaps with `[ancient times]`. The first is retained because this is the leftmost occurrence and `[ancient times]` is eliminated. The following occurrence of `[times a]` is no longer in conflict with `[ancient times]` and can therefore appear in the result:

```
...Don, there extended [in ancient] [times a] large forest...
```

The rule of priority of the leftmost match is applied only when the text is modified, be it during preprocessing or after the application of a syntactic graph (cf. section 6.7.3).

6.6.4 Priority of the longest match

During the application of a syntactic graph it is possible to choose if the priority should be given to the shortest or the longest sequences or if all sequences should be retained. During

preprocessing, the priority is always given to the longest sequences.

6.6.5 Transducer outputs with variables

As we have seen in section 5.2.6, it is possible to use variables to store the text that has been analyzed by a grammar. These variables can be used in preprocessing graphs and in syntactic graphs.

You have to give names to the variables you use. These names can contain non-accentuated lower-case and upper-case letters between `A` and `Z`, digits and the character _ (underscore).

In order to define the end of the zone that is stored in a variable, you have to create a box that contains the name of the variable enclosed in the characters `$` and `(` (`$` and `)` for the end of a variable). In order to use a variable in a transducer output, its name must be preceded by the character `$` (cf. figure 6.25).

Variables are global. This means that you can define a variable in a graph and reference it in another as is illustrated in the graphs of figure 6.25.

If graph `TitleName` is applied in MERGE mode to the text *Ivanhoe*, the concordance in figure 6.26 is obtained.

Outputs with variables can be used to move groups of words. In fact, the application of a transducer in REPLACE mode inserts only the produced sequences into the text. In order to inverse two groups of words it is sufficient to store them into variables and produce an output with these variables in the desired order. Thus, the application of the transducer in figure 6.27 in REPLACE mode to the text *Ivanhoe* results in the concordance of figure 6.28.

If the beginning or the end of variable is malformed (end of a variable before its beginning or absence of the beginning or end of a variable), it will be ignored during the emission of outputs.

There is no limit to the number of possible variables.

The variables can be nested and even overlap as is shown in figure 6.29.

6.7 Applying graphs to texts

This section only applies to syntactic graphs.

6.7.1 Configuration of the search

In order to apply a graph to a text, you open the text, then click on "Locate Pattern..." in the "Text" menu, or press <Ctrl+L>. You can then configure your search in the window shown in figure 6.30.

In the "Locate pattern in the form of" field, choose "Graph" and select your graph by clicking on the "Set" button. You can choose a graph in `.grf` format (Unicode Graphs) or a compiled graph in `.fst2` format (Unicode Compiled Graphs). If your graph is in `.grf` format, Unitex will compile it automatically before starting the search.

The "Index" field allows to select the recognition mode.

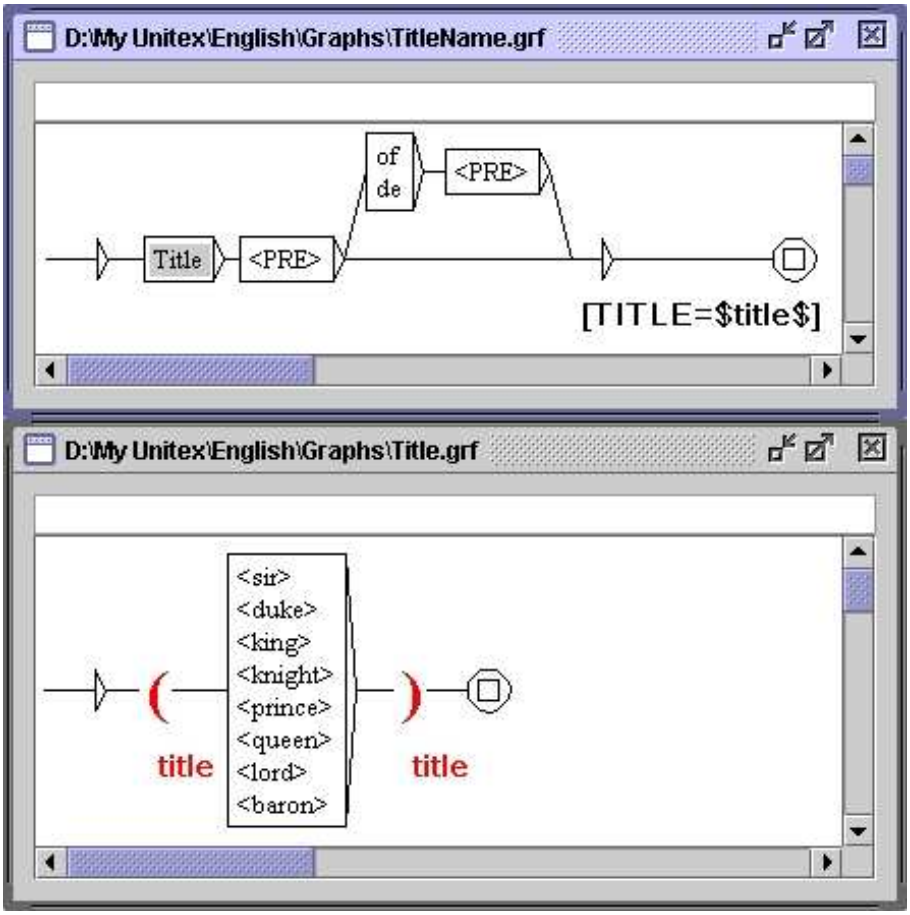


Figure 6.25: Definition of a variable in a subgraph

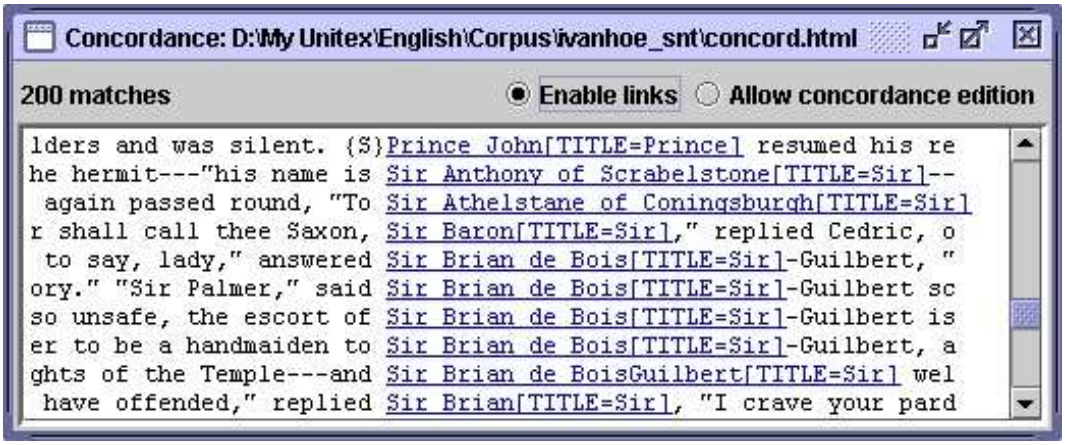


Figure 6.26: Concordance obtained by application of graph `TitleName`

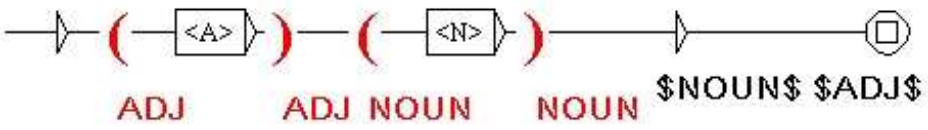


Figure 6.27: Inversion of words using two variables

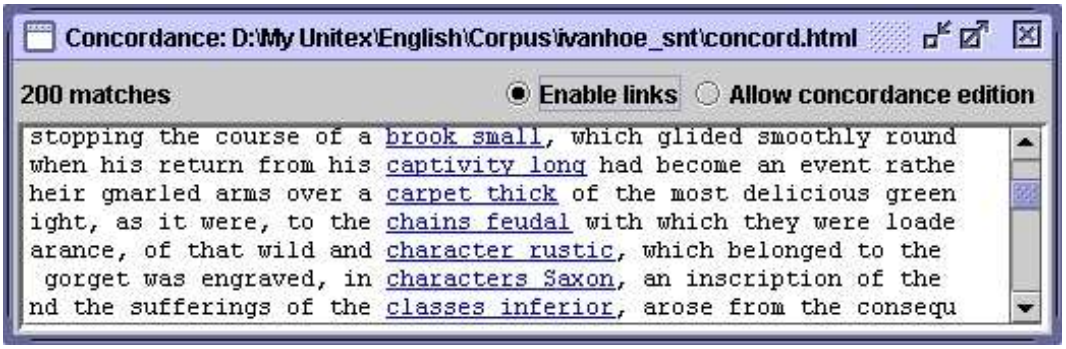


Figure 6.28: Result of the application of the transducer in figure 6.27

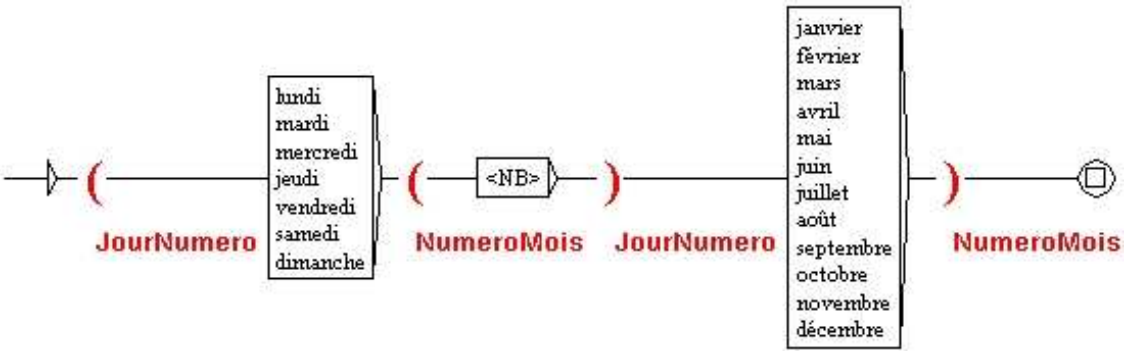


Figure 6.29: Overlapping of variables

- "Shortest matches" : give precedence to the shortest matches;
- "Longest matches" : give precedence to the longest sequences. This is the default mode;
- "All matches" : give out all recognized sequences.

The "Search limitation" field allows you to limit the search to a certain number of occurrences. By default, the search is limited to the 200 first occurrences.

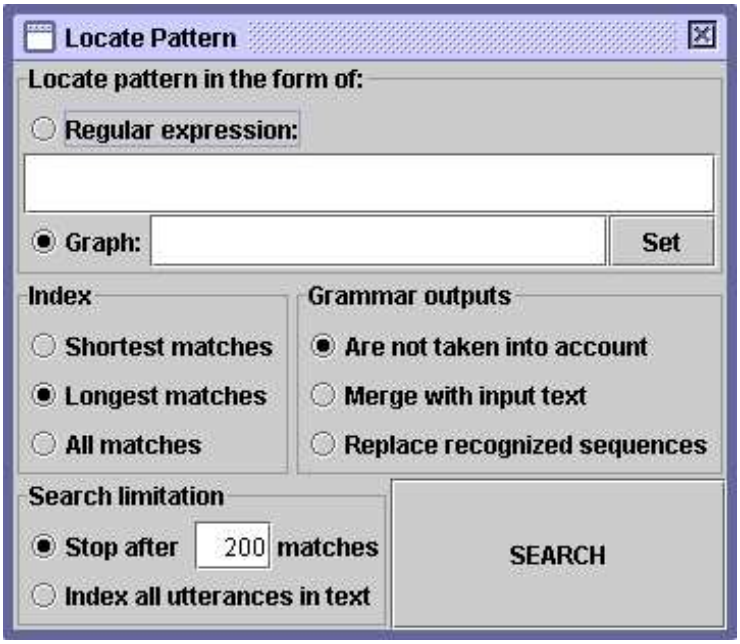


Figure 6.30: Window for pattern search

The "Grammar outputs" field concerns transducers. The "Merge with input text" mode allows you to insert the output sequences. The "Replace recognized sequences" mode allows you to replace the recognized sequences with the produced sequences. The third mode ignores all outputs. This latter mode is used by default.

After you have selected the parameters, click on "SEARCH" to start the search.

6.7.2 Concordance

The result of a search is an index file that contains the positions of all encountered occurrences. The window of figure 6.31 lets you choose whether to construct a concordance or modify the text.

In order to display a concordance, you have to click on the "Build concordance" button. You can parameterize the size of left and right contexts in characters. You can also choose the sorting mode that will be applied to the lines of the concordance in the "Sort According to" menu. For further details on the parameters of concordance construction, refer to section 4.8.2.

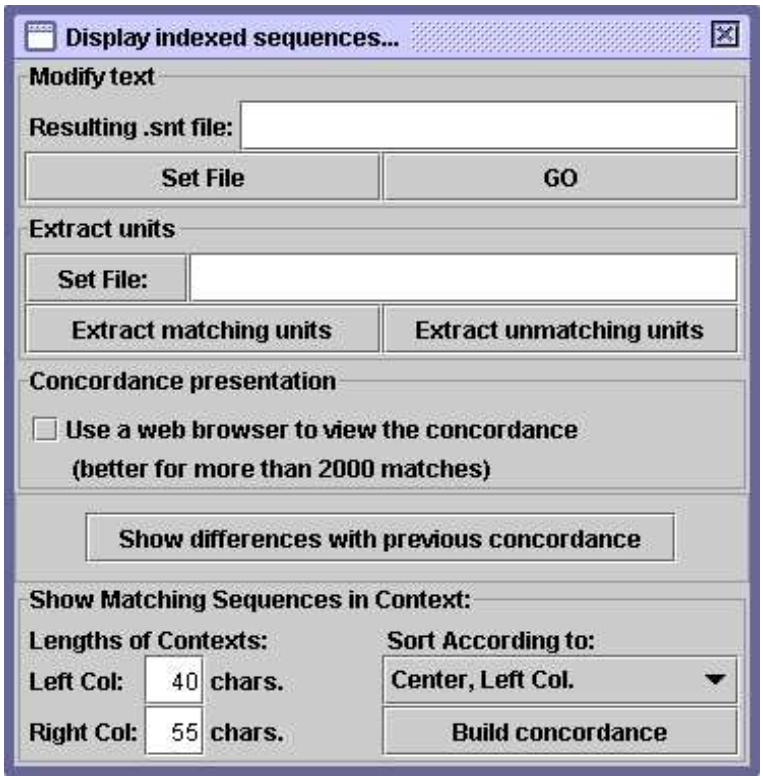


Figure 6.31: Configuration for displaying the encountered occurrences

The concordance is produced in the form of an HTML file. You can parameterize Unitex so that the concordances can be read using a web browser (cf. section 4.8.2).

If you display the concordances with the window provided by Unitex, you can access a recognized sequence in the text by clicking on the occurrence. If the text window is not iconified and the text is not too long to be displayed, you see the selected sequence appear (cf. figure 6.32).

Furthermore, if the text automaton has been constructed, and if the corresponding window is not iconified, clicking on an occurrence selects the automaton of the sentence that contains this occurrence.

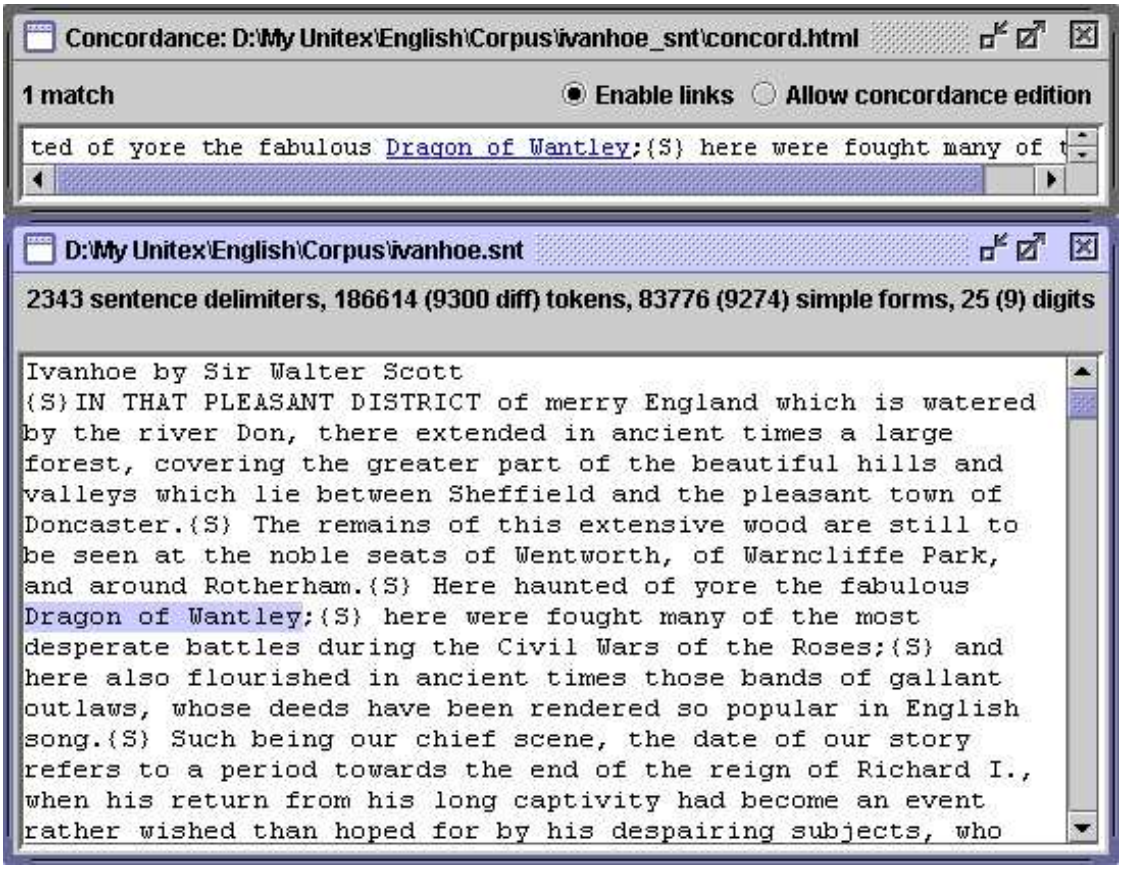


Figure 6.32: Selection of an occurrence in the text

6.7.3 Modification of the text

You can choose to modify the text instead of constructing a concordance. In order to do that type a file name in the field "Modify text" in the window of figure 6.31. This file has to have the extension `.txt`.

If you want to modify the current text, you have to choose the corresponding `.txt` file. If you choose another file name, the current text will not be affected. Click on the "GO" button to start the modification of the text. The precedence rules that are applied during these operations are described in section 3.6.2.

After this operation the resulting file is a copy of the text in which all transducer outputs have been taken into account. Normalization operations and splitting into lexical units are automatically applied to this text file. The existing text dictionaries are not modified. Thus, if you have chosen to modify the current text, the modifications will be effective immediately. You can then start new searches on the text.

ATTENTION: if you have chosen to apply your graph ignoring the transducer outputs,

all occurrences will be erased from the text.

6.7.4 Extracting occurrences

To extract from a text all sentences containing matches, set the name of your output text file using the "Set File" button in the "Extract units" frame (figure 6.31). Then, click on "Extract matching units". Instead, by clicking on "Extract unmatching units" all sentences not containing any matches will be extracted.

6.7.5 Comparing concordances

With the "Show differences with previous concordance" option, you can compare the current concordance with the previous one. The `ConcorDiff` program builds both concordances according to text order and compares them line by line. The result is an HTML page that presents results in two columns. A blue line indicates that an utterance is common to the two concordances. A red line indicates that a match is common to both concordances but with different range, i.e. the two matches only overlap partially. A green line indicates an utterance that appears in only one concordance. Figure 6.33 gives an example.

NOTE: you cannot click on utterances in a concordance comparison.
If you have no previous concordance the button is deactivated.

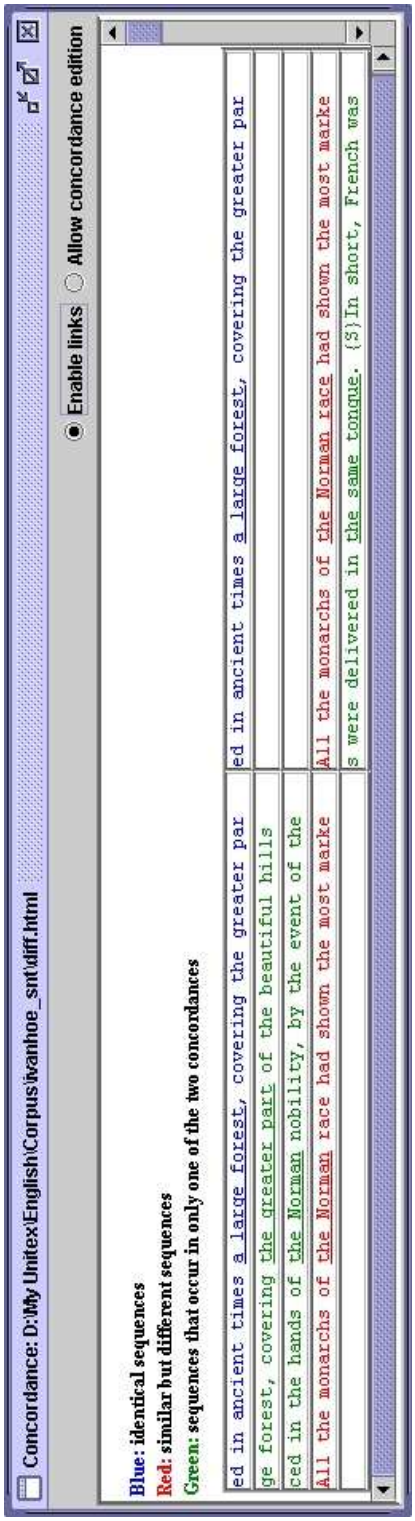


Figure 6.33: Example of a concordance comparison

Chapter 7

Text automata

Natural languages contain much lexical ambiguity. The text automaton is an effective and visual means of representing such ambiguity. Each sentence of a text is represented by an automaton whose paths represent all possible interpretations.

This chapter presents the concept of text automata, the details of their construction and the operations that can be applied, in particular ambiguity removal with ELAG ([35]). It is not possible at the moment to search the text automaton for patterns.

7.1 Displaying text automata

The text automaton can express all possible lexical interpretations of the words. These different interpretations are the different entries presented in the dictionary of the text. Figure 7.1 shows the automaton of the fourth sentence of the text *Ivanhoe*.

You can see in figure 7.1 that the word `Here` has three interpretations here (adjective, adverb and noun), `haunted` two (adjective and verb), etc. All the possible combinations are expressed because each interpretation of each word is connected to all the interpretations of the following and preceding words.

In case of an overlap between a compound word and a sequence of simple words, the automaton contains a path that is labeled by the compound word, parallel to the paths that express the combinations of simple words. This is illustrated in figure 7.2, where the compound word `courts of law` overlaps with a combination of simple words.

By construction, the automaton of the text doesn't contain any loops. One says that the text automaton is *acyclic*.

NOTE: the term "text automaton" is an abuse of language. In fact, there is an automaton for each sentence of the text. Therefore, the combination of all these automata corresponds to the automaton of the text. We use the term text automaton even if this object is not manipulated as a global automaton for practical reasons.

7.2 Construction

In order to construct the text automaton, open the text, then click on "Construct FST-Text..." in the menu "Text". One should first split the text at sentence boundaries and apply the dic-

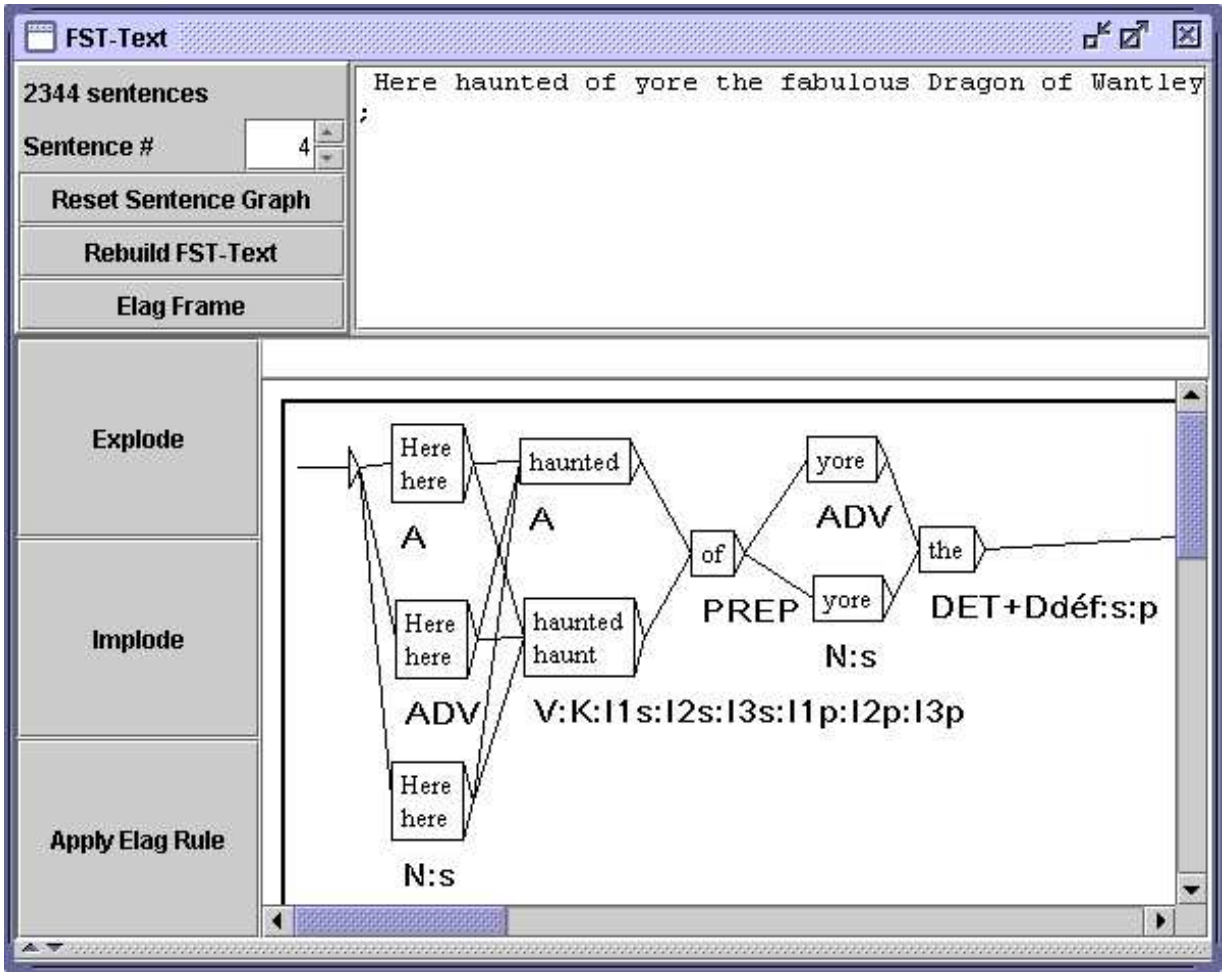


Figure 7.1: Example of the automaton of a sentence

tionaries. If sentence boundary detection is not applied, the construction program will split the text arbitrarily in sequences of 2000 lexical units instead of constructing one automaton per sentence. If no dictionaries are applied, the sentence automaton that you obtain will consist of only one path made up of unknown words.

7.2.1 Construction rules for text automata

The sentence automata are constructed from text dictionaries. The resulting degree of ambiguity is therefore directly linked to the granularity of the descriptions of the used dictionaries. From the sentence automaton in figure 7.3, you can conclude that the word `which` has been coded twice as a determiner in two subcategories of the category `DET`. This granularity of descriptions will not be of any use if you are not interested in the grammatical category of this word. It is therefore necessary to adapt the granularity of the dictionaries to

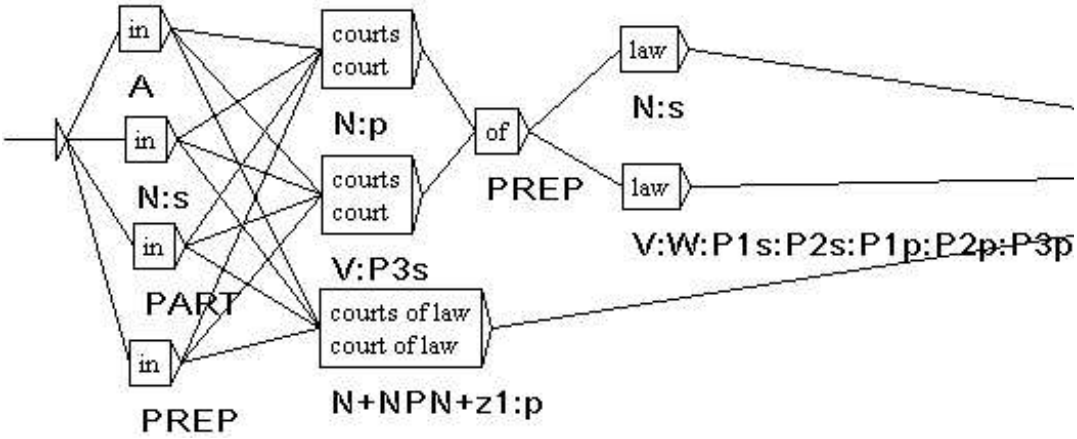


Figure 7.2: Overlap between a compound word and a combination of simple words.

the intended use.

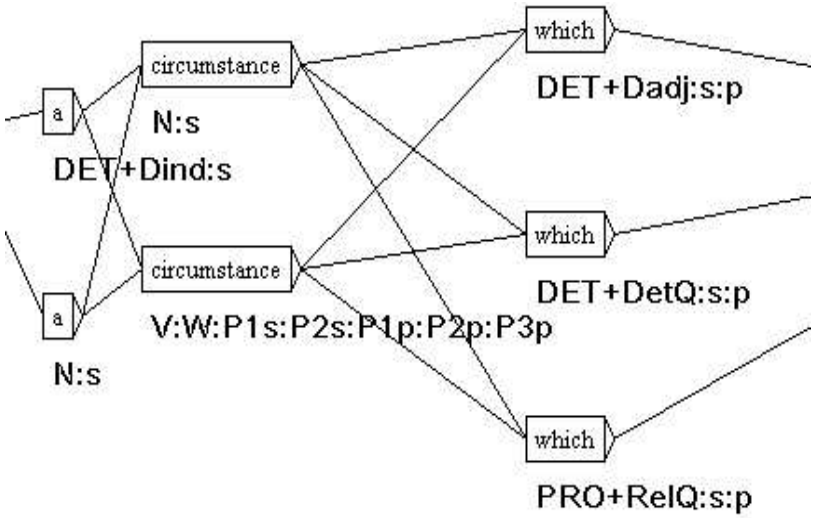


Figure 7.3: Double entry for `which` as a determiner

For each lexical unit of the sentence, Unitex searches the dictionary of the simple words of the text for all possible interpretations. Afterwards all lexical units that have an interpretation in the dictionary of the compound words of the text are sought. All the combinations of their interpretations constitute the sentence automaton.

NOTE: If the text contains lexical labels (*e.g.* `{out of date,.A+z1}`), these labels are reproduced identically in the automaton without trying to decompose the sequences which they represent.

In each box, the first line contains the inflected form found in the text, and the second line contains the canonical form if it is different. The other information is coded below the box. (cf. section 7.4.1).

The spaces that separate the lexical units are not copied into the automaton except for the spaces inside compound words.

The case of lexical units is retained. For example, if the word `Here` is encountered, the capital letter is preserved (cf. figure 7.1). This choice allows you to keep this information during the transition to the text automaton, which could be useful for applications where case is important as for recognition of proper names.

7.2.2 Normalization of ambiguous forms

During construction of the automaton, it is possible to effect a normalization of ambiguous forms by applying a normalization grammar. This grammar has to be called `Norm.fst2` and must be placed in your personal folder, in the subfolder `/Graphs/Normalization` of the desired language. The normalization grammars for ambiguous forms are described in section 6.1.3.

If a sequence of the text is recognized by the normalization grammar, all the interpretations that are described by the grammar are inserted into the text automaton. Figure 7.4 shows the part of the grammar used for the ambiguity of the sequence `l'` in French.

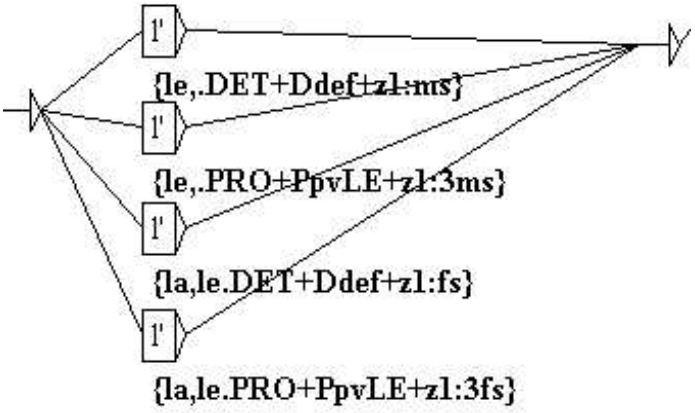


Figure 7.4: Normalization of the sequence `l'`

If this grammar is applied to a French sentence containing the sequence `l'`, a sentence automaton that is similar to the one in figure 7.5 is obtained.

You can see that the four rules for rewriting the sequence `l'` have been applied, which has added four labels to the automaton. These labels are concurrent with the two preexisting paths for the sequence `l'`. The normalization at the time of the construction of the automaton allows you to add paths to the automaton but not to erase paths. The disambiguation functionality allows you to eliminate the paths that have become superfluous.

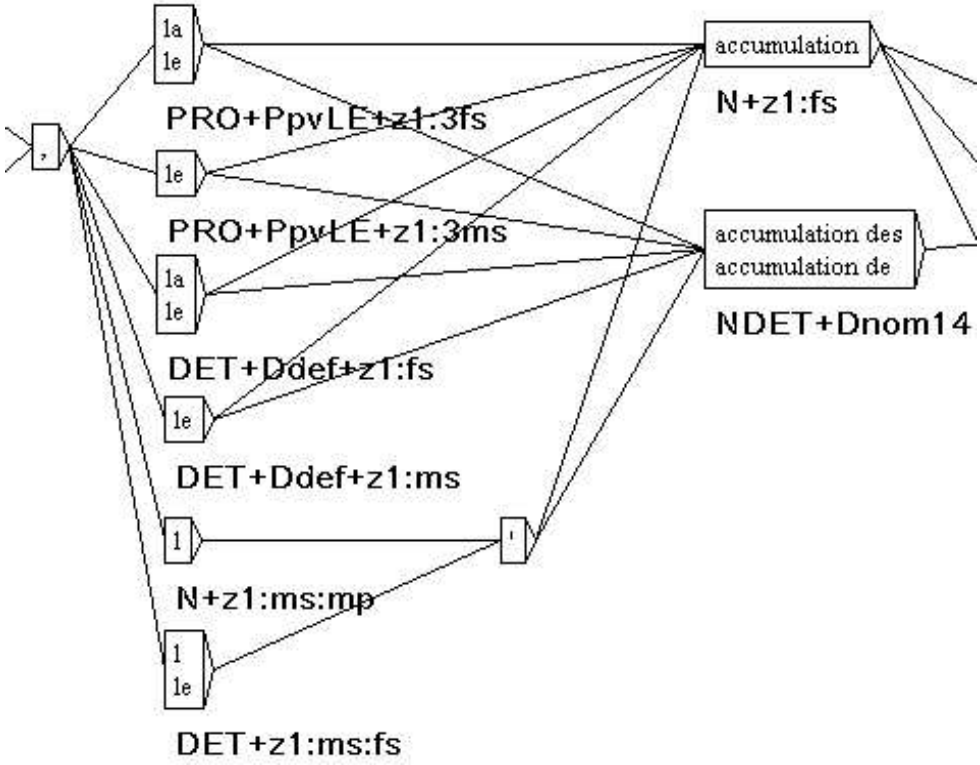


Figure 7.5: Automaton that has been normalized with the grammar of figure 7.4

7.2.3 Normalization of clitical pronouns in Portuguese

In Portuguese, verbs in the future tense and in the conditional can be modified by the insertion of one or two clitical pronouns between the root and the suffix of the verb. For example, the sequence *dir-me-ão* (*they will tell me*), corresponds to the complete verbal form *dirão*, associated with the pronoun *me*. In order to be able to manipulate this rewritten form, it is necessary to introduce it into the text automaton in parallel to the original form. Thus, the user can search one or the other form. The figures 7.6 and 7.7 show the automaton of a sentence after normalization of the clitics.

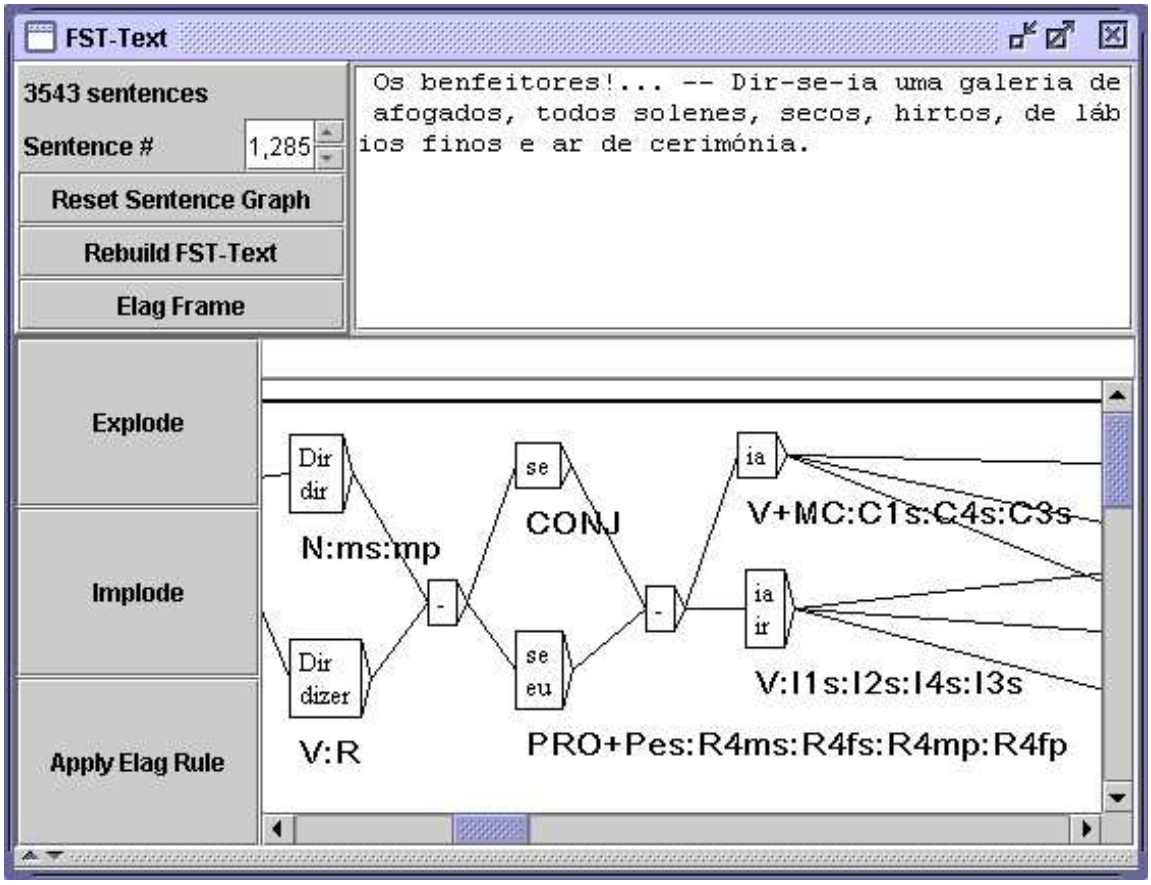


Figure 7.6: Non-normalized phrase automaton

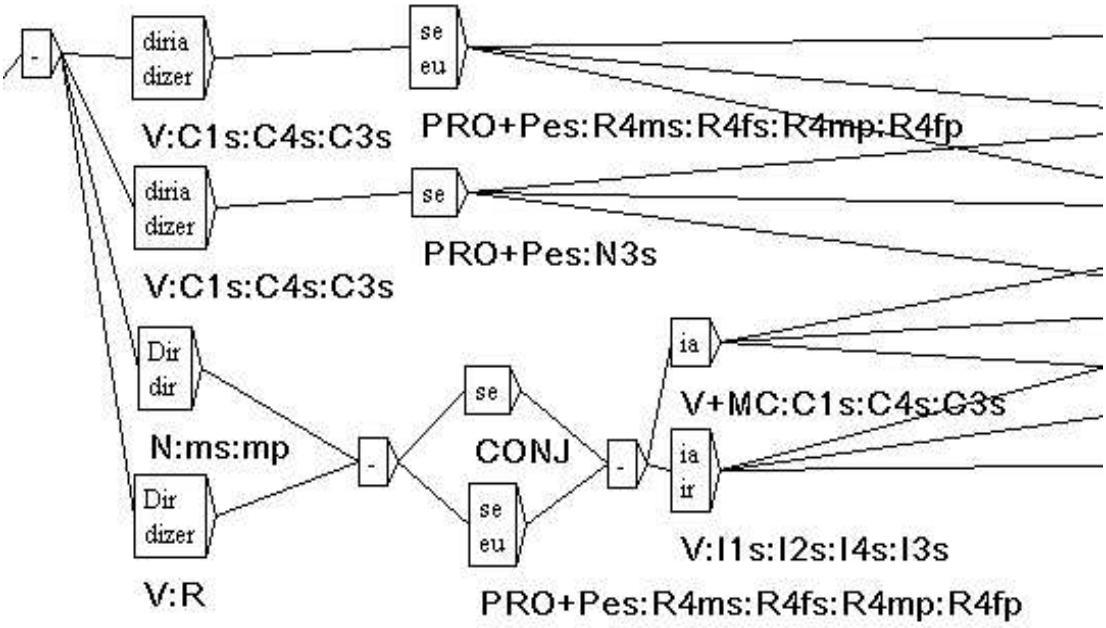


Figure 7.7: Normalized phrase automaton

The program `Reconstrucao` allows you to construct a normalization grammar for these forms for each text dynamically. The grammar thus produced can then be used for normalizing the text automaton. The configuration window of the automaton construction suggests an option "Build clitic normalization grammar" (cf. figure 7.10). This option automatically starts the construction of the normalization grammar, which is then used to construct the text automaton, if you have selected the option "Apply the Normalization grammar".

7.2.4 Keeping the best paths

An unknown word can perturb the text automaton by overlapping with a completely labeled sequence. Thus, in the automaton of figure 7.8, it can be seen that the adverb `aujourd'hui` overlaps with the unknown word `aujourd`, followed by an apostrophe and the past participle of the verb `huir`.

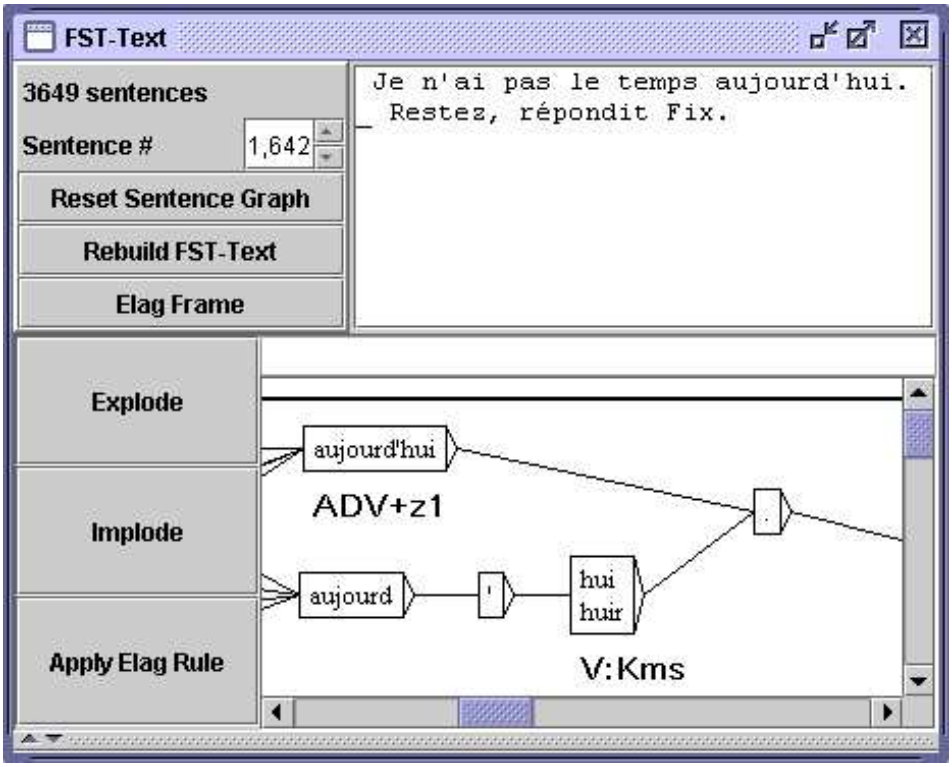


Figure 7.8: Ambiguity due to a sentence conaining an unkown word

This phenomenon can also take place in the treatment of certain Asian languages like Thaï. When words are not delimited, there is no other solution than to consider all possible combinations, which causes the creation of numerous paths carrying unknown words that are mixed with the labeled paths. Figure 7.9 shows an example of such an automaton of a Thaï sentence.

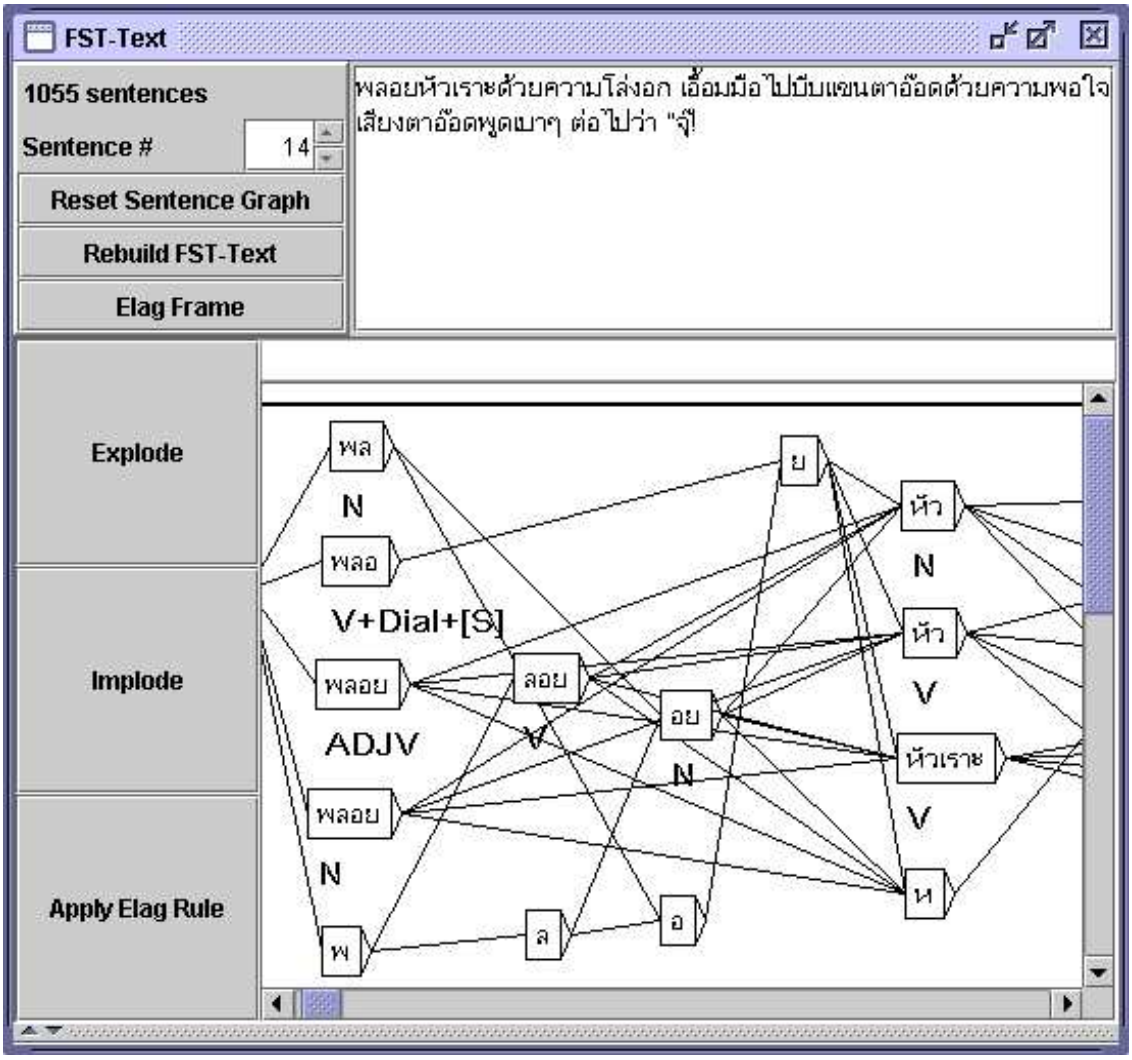


Figure 7.9: Automaton of a thaï sentence

It is possible to suppress parasite paths. You have to select the option "Clean Text FST" in the configuration window for the construction of the text automaton (cf. figure 7.10). This option indicates to the automaton construction program that it should clean up each sentence automaton.

This cleaning is carried out according to the following principle: if several paths are concurrent in the automaton, the program keeps those that contain the fewest unknown words. For instance, the compound adverb `aujourd'hui` is preferred to the sequence made of `aujourd` followed by a quote and `hui`, because `aujourd` is an unknown word, while the compound adverb path does not contain any unknown word.

Figure 7.11 shows the automaton of figure 7.9 after cleaning.

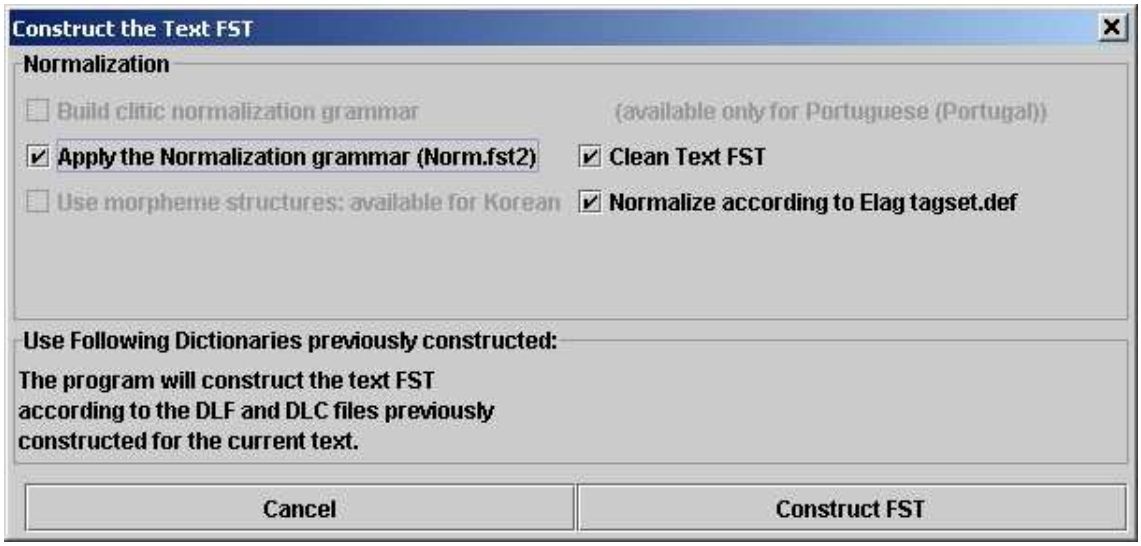


Figure 7.10: Configuration of the construction of the text automaton

7.3 Resolving Lexical Ambiguities with ELAG

The ELAG program allows for applying grammars for ambiguity removal to the text automaton. This powerful mechanism makes it possible to write rules on independently from already existing rules. This chapter briefly presents the grammar formalism used by ELAG and describes how the program works. For more details, the reader may refer to [3] and [35].

7.3.1 Grammars For Resolving Ambiguities

The grammars used by ELAG have a special syntax. They consist of two parts which we call the if and $then$ parts. The if part of an ELAG grammar is divided in two parts which are divided by a box containing the <!> symbol. The $then$ part is divided the same way using the <=> symbol. The meaning of a grammar is the following: In the text automaton, if a path of the if part is recognized, then it must also be recognized by the $then$ part of the grammar, or it will be withdrawn from the text automaton.

Figure 7.12 shows an example of a grammar. The if part recognizes a verb in the 2nd person singular followed by a dash and tu, either as a pronoun, or as a past participle of the verb $taire$. The $then$ part imposes that tu is then regarded as a pronoun. Figure 7.13 shows the result of the application of this grammar on the sentence "*Feras-tu cela bientôt ?*". One can see in the automaton at the bottom that the path corresponding to tu past participle was eliminated.

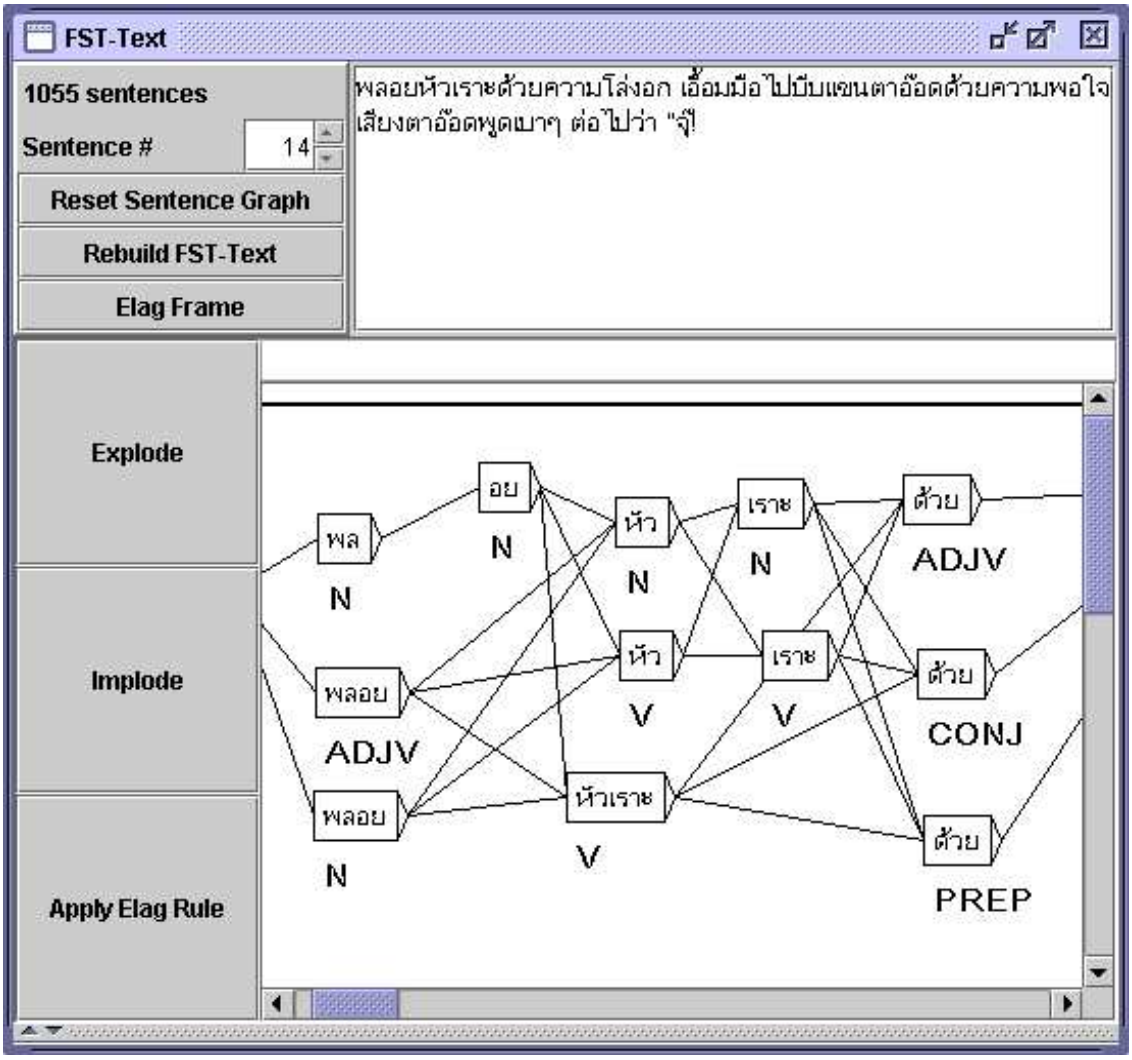


Figure 7.11: Automaton of figure 7.9 after cleaning

Synchronization point

The if and $then$ parts of an ELAG grammar are divided into two parts by <!> in the if part, and <=> in the $then$ part. These symbols form a $point\ of\ synchronization$. This makes it possible to write rules in which the if and $then$ constraints are not necessarily aligned, as it is the case for example in figure 7.14. This grammar is interpreted in the following way: if a dash is found followed by il, $elle$ or on, then this dash must be preceded by a verb, possibly followed by $-t$. So, if one considers the sentence of the figure 7.15 beggining with *Est-il*, one can see that all non-verb interpretations of Est were removed.

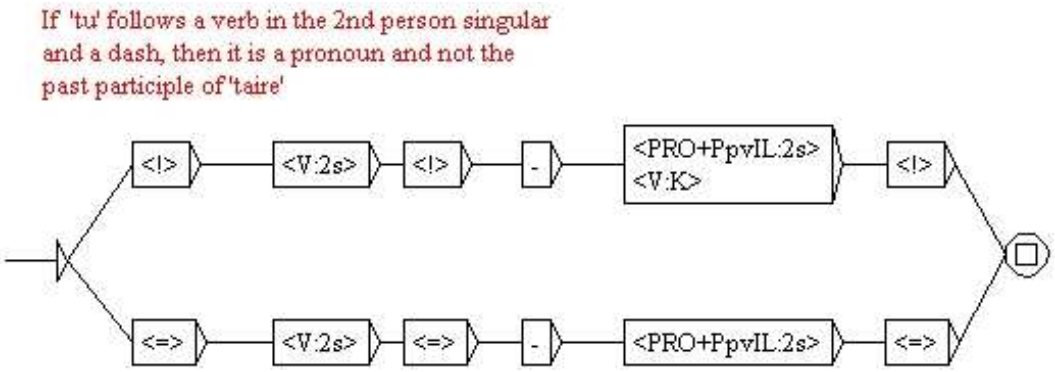


Figure 7.12: Example of an ELAG grammar

7.3.2 Compiling ELAG Grammars

Before an ELAG grammar can be applied to a text automaton, the grammar must be compiled in a `.rul` file. This operation is carried out via the "Elag Rules" command in the "Text" menu, which opens the windows shown in figure 7.16.

If the frame on the right already contains grammars which you don't wish to use, you can withdraw them with the << button. Then select your grammar(s) in the file explorer located in the left frame, and click on the >> button to add them to the list in the right frame. Then click on the *compile* button. This will launch the `ElagComp` program which will compile the selected grammars and create a file named *elag.rul* by default.

If you have selected grammars in the right frame, you can search patterns whith them by clicking on the *locate* button. This opens the window "Locate Pattern" and automatically enters a graph name ending with `-conc.fst2`. This graphs corresponds to the if part of the grammar. You can thus obtain the occurrences of the text to which the grammar will apply.

NOTE: The `-conc.fst2` file used to locate the *then* part of a grammar is generated at the time when ELAG grammars are compiled by means of the *compile* button. It is thus necessary initially to have your grammar compiled before searching using the *locate* button.

7.3.3 Resolving Ambiguities

Once you have compiled your grammar into an `elag.rul` file, you can apply it to a text automaton. In the text automaton window, click on the *elag* button. A dialog box will appear which asks for the the `.rul` file to be used (see figure 7.17). The default file is `elag.rul`. This will launch the `Elag` program which will resolve the ambiguity.

Once the program has finished you can view the resulting automaton by clicking on the *Elag Frame* button. As you can see in figure 7.18, the windows is separated into two parts: The original text automaton can be seen on the top, and the result at the bottom.

Don't be surprised if the automaton shown at the bottom seems more complicated. This results from the fact that factorized lexical entries[1] were exploded in order to treat each

[1]Entries which gather several different inflectional interpretations, such as for example:

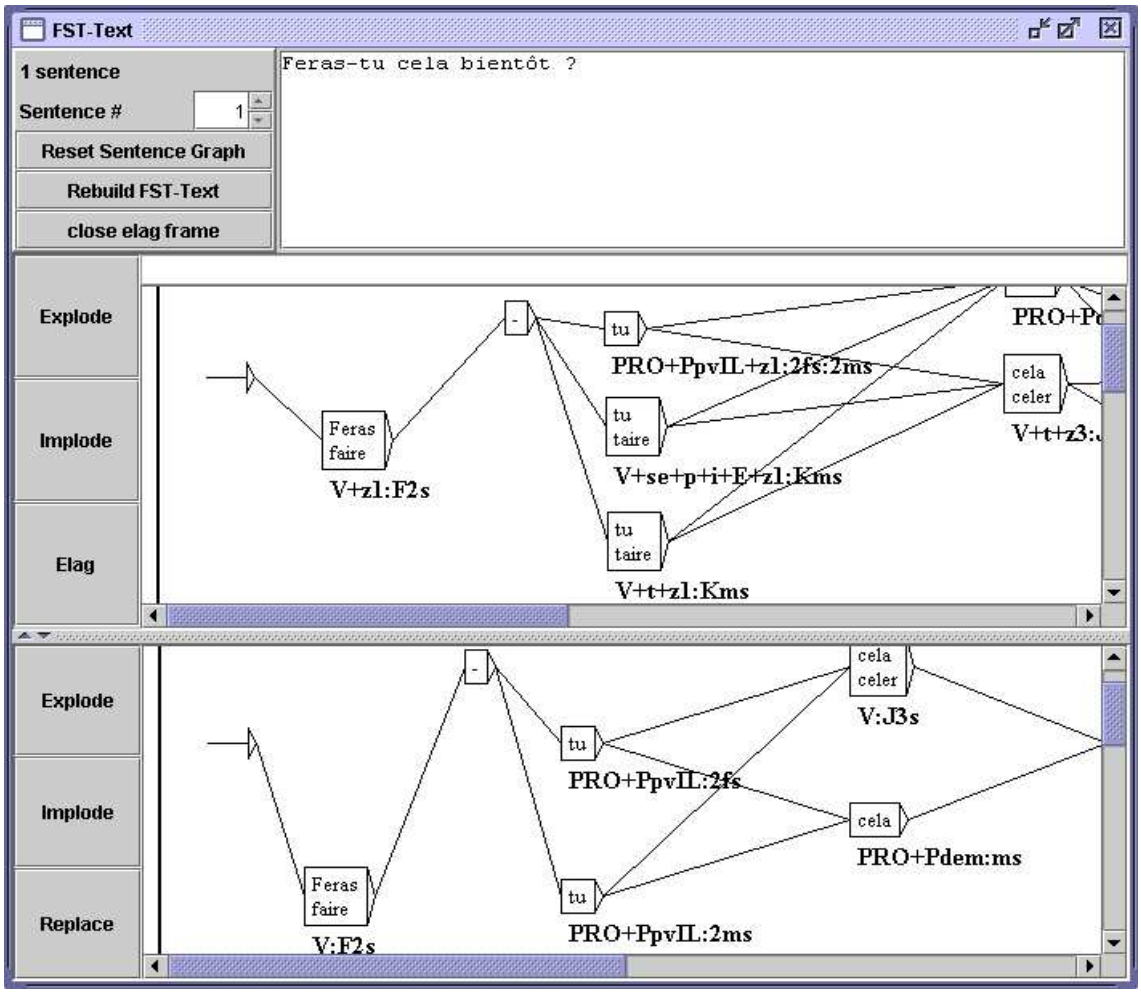


Figure 7.13: Result of applying the grammar in figure 7.12

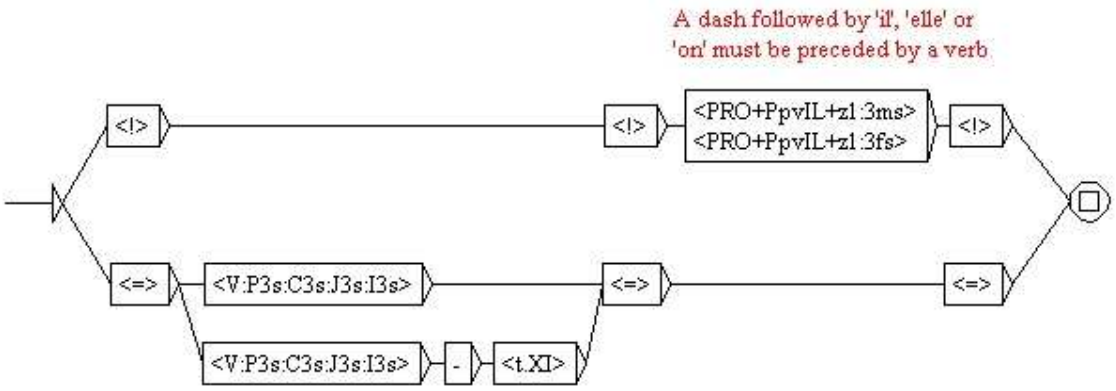


Figure 7.14: Use of the synchronization point

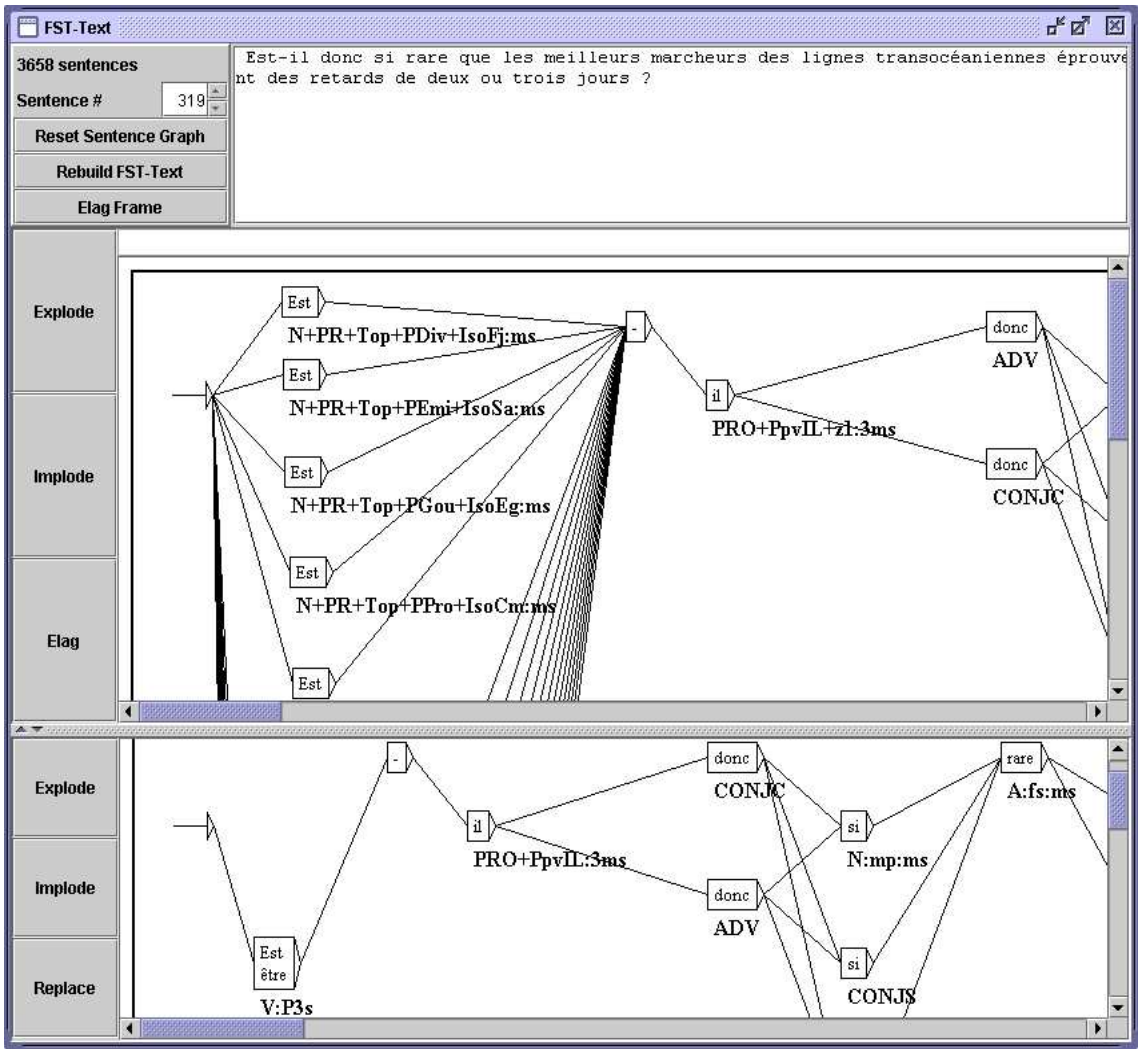


Figure 7.15: Result of the application of the grammar in figure 7.14

inflectional interpretation separately. To refactorize these entries, click on the *implode* button. Clicking on the button *explode* shows you an exploded view of the text automaton.

If you click on the *replace* button, the resulting automaton will become the new text automaton. Thus, if you use other grammars, they will apply to the already partially disambiguated automaton, which makes it possible to accumulate the effects of several grammars.

7.3.4 Grammar collections

It is possible to gather several ELAG grammars into a grammar collection in order to compile and apply them in one step. The sets of ELAG grammars are described in `.lst` files. They

`{se,.PRO+PpvLE:3ms:3fs:3mp:3fp}`.

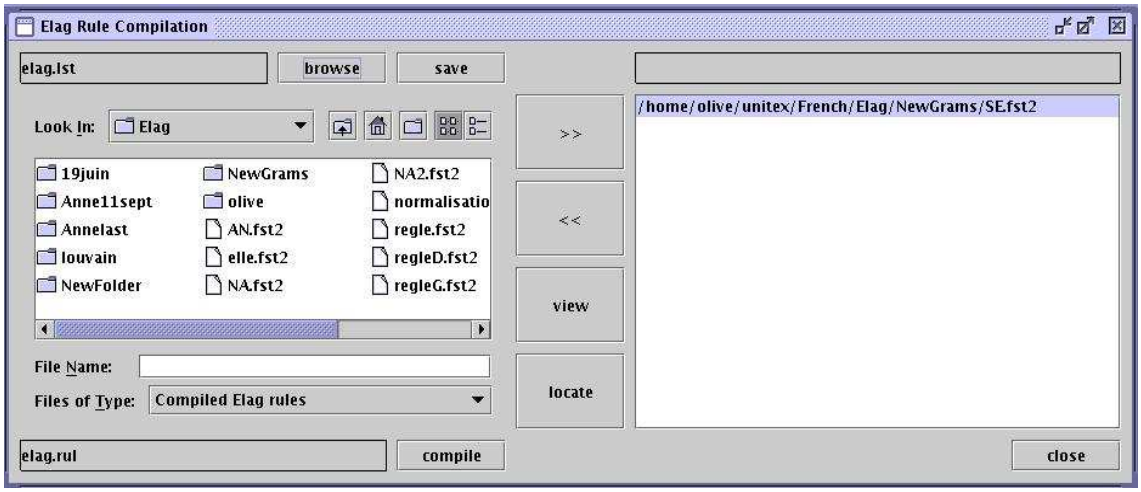


Figure 7.16: ELAG grammars compilation frame

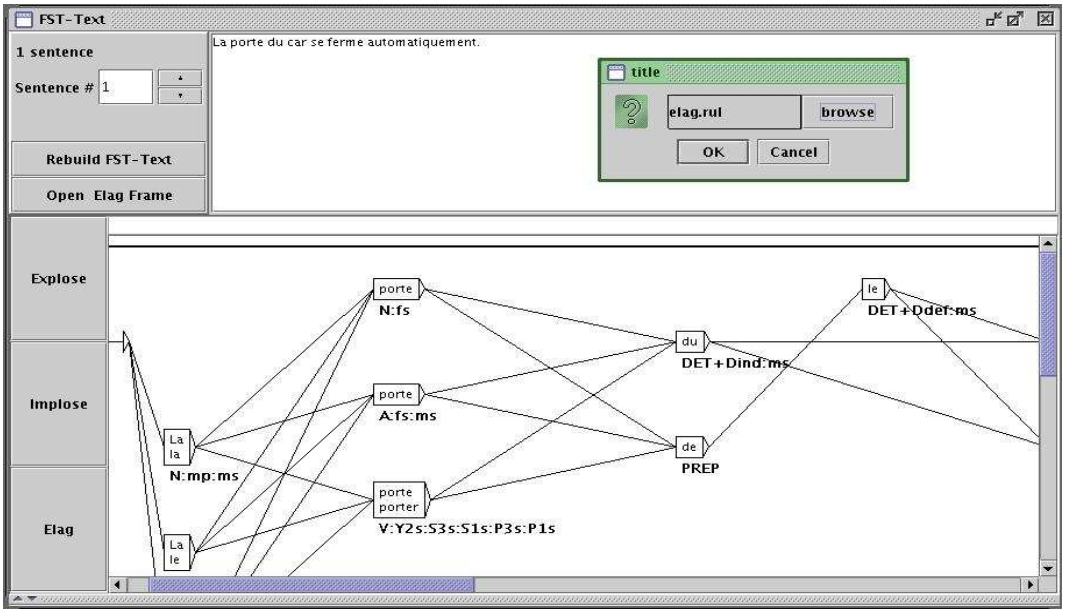


Figure 7.17: Text automaton frame

are managed through the window for compiling ELAG grammars (figure 7.16). The label on the top left indicates the name of the current collection, by default `elag.lst`. The contents of this collection are displayed in the right part of the window.

To modify the name of the collection, click on the *browse* button. In the dialog box that appears, enter the `.lst` file name for the collection.

To add a grammar to the collection, select it in the file explorer in the left frame, and click on the >> button. Once you selected all your grammars, compile them by clicking on the

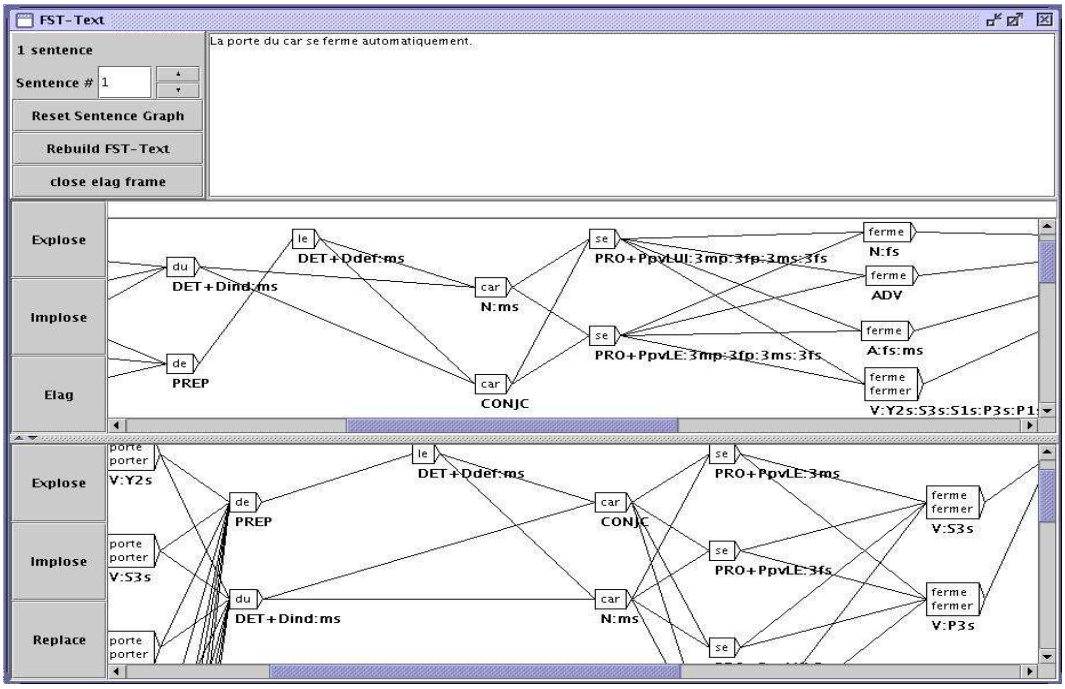


Figure 7.18: Splitted text automaton frame

compile button. This will create a `.rul` file bearing the name indicated at the bottom right (the name of the file is obtained by replacing the `.lst` by the `.rul` extension).

You can now apply your grammar collection. As explained above, click on the *elag* button in the text automaton window. When the dialog asks for the `.rul` file to use, click on the *browse* button and select your collection. The resulting automaton is identical to that which would have been obtained by applying each grammar successively.

7.3.5 Window For ELAG Processing

At the time of disambiguation, the `Elag` program is launched in a processing window which displays the messages printed by the program during its execution.

For example, when the text automaton contains symbols which do not correspond to the set of ELAG labels (see the following section), a message indicates the nature of the error. In the same way, when a sentence is rejected (all possible analyses were eliminated by grammars), a message indicates the number of the sentence. That makes it possible to locate the source of the problems quickly.

Evaluation of ambiguity removal

The evaluation of the rate of ambiguity is not based solely on the average number of interpretations per word. In order to get a more representative measure, the system also takes into account the various combinations of words. While instances of ambiguities are

resolved, the `elag` program calculates the number of possible analyses in the text automaton before and after the modification (which corresponds to the number of possible paths through the automaton). Based on this value, the program computes the average ambiguity by sentence and word. It is this last measure which is used to represent the rate of ambiguity of the text, because it does not vary with the size of the corpus, nor with the number of sentences within. The formula applied is:

$$\textit{lexical ambiguity rate} = exp^{\frac{log(number-of-paths)}{text-length}}$$

The relationship between the rate of ambiguities before and after applying the grammars gives a measure of their efficiency. All this information is displayed in the ELAG processing window.

7.3.6 Description Of The Tag Sets

The `Elag` and `ElagComp` programs indexExternal programs!`Elag` require a formal description of the tag set of the dictionaries used. This description consists essentially of an enumeration of all the parts of speech present in the dictionaries, with, for each of them, the list of syntactic and inflectional codes compatible with it, and a description of their possible combinations. This information is described in the file named `tagset.def`.

`tagset.def` file

Here is an extract of the `tagset.def` file used for French.

```
NAME français

POS ADV
.

POS PRO
inflex:
  pers   = 1 2 3
  genre  = m f
  nombre = s p
discr:
  subcat = Pind Pdem PpvIL PpvLUI PpvLE Ton PpvPR PronQ Dnom Pposs1s...
complete:
Pind     <genre> <nombre>
Pdem     <genre> <nombre>
Pposs1s  <genre> <nombre>
Pposs1p  <genre> <nombre>
```

```
Pposs2s  <genre> <nombre>
Pposs2p  <genre> <nombre>
Pposs3s  <genre> <nombre>
Pposs3p  <genre> <nombre>
PpvIL    <genre> <nombre> <pers>
PpvLE    <genre> <nombre> <pers>
PpvLUI   <genre> <nombre> <pers>      #
Ton      <genre> <nombre> <pers>      # lui, elle, moi
PpvPR                                 # en y
PronQ                                 # où qui que quoi
Dnom                                  # rien
.

POS A ## adjectifs
inflex:
  genre  = m f
  nombre = s p
cat:
  gauche = g
  droite = d
complete:
<genre> <nombre>
_  # pour {de bonne humeur,.A}, {au bord des larmes,.A} par exemple
.


POS V
inflex:
  temps  = C F I J K P S T W Y G X
  pers   = 1 2 3
  genre  = m f
  nombre = s p
complete:
W
G
C <pers> <nombre>
F <pers> <nombre>
I <pers> <nombre>
J <pers> <nombre>
P <pers> <nombre>
S <pers> <nombre>
T <pers> <nombre>
X 1 s   # eussé dussé puissé fussé (-je)
Y 1 p
Y 2 <nombre>
```

```
K <genre> <nombre>
.
```

The # symbol indicates that the remainder of the line is a comment. A comment can appear at any place in the file. The file always starts with the word `NAME`, followed by an identifier (`français`, for example). This is followed by the `POS` sections for each part of speech. Each section describes the structure of the lexical tags of the lexical entries belonging to the part of speech concerned. Each section is composed of 4 parts which are all optional:

- `inflex`: this part enumerates the inflectional codes belonging to the grammatical category. For example, the codes `1,2,3` which indicate the person of the entry are relevant for pronouns but not for adjectives. Each line describes an inflexional attribute (gender, time, etc.) and is made up of the attribute name, followed by the = character and the values which it can take; For example, the following line declares an attribute $pers$ being able to taking the values 1, 2 or 3:

  ```
  pers = 1 2 3
  ```

- `cat`: this part declares the syntactic and semantic attributes which can be assigned to the entries belonging to the part of speech concerned. Each line describes an attribute and the values which it can take. The codes declared for the same attribute must be exclusive. In other words, an entry cannot take more than one value for the same attribute.

  On the other hand, all the tags in a given part of speech don't necessarily take values for all the attribute of the part of speech. For example, to define the attribute `niveau_de_langue` which can take the values `z1`, `z2` and `z3`, the following line can be written:

  ```
  niveau_de_langue = z1 z2 z3
  ```

  but this attribute is not necessarily present in all words.

- `discr`: this part consists of a declaration of a unique attribute. The syntax is the same as in the `cat` part and the attribute described here must not be repeated there. This part allows for dividing the grammatical category in *discriminating* sub categories in which the entries have similar inflectional attributes. For pronouns for example, a person feature is assigned to entries that are part of the personal pronoun sub category but not to relative pronouns. These dependencies are described in the `complete` part;

- `complete`: this part describes the inflectional part of the tags of the words in the current part of speech. Each line describes a valid combination of inflectional codes by their discriminating sub category (if such a category was declared). If an attribute name is specified in angle brackets (< and >), this signifies that any value of this attribute may occur. It is possible as well to declare that an entry does not take any inflexional feature by means of a line containing only the _ character (underscore). So for example, if we consider that the following lines extracted from the section describing the verbs:

```
W
K <genre> <nombre>
```

They make it possible to declare that verbs in the infinitive (indicated by the `W` code) do not have other inflectional features while the forms in the past participle (`K` code) are also assigned a gender and a number.

Description of the inflectional codes

The principal function of the `discr` part is to divide a part of speech into subcategories having similar inflectional behavior. These subcategories are then used to facilitate writing the `complete` part.

For the legibility of the ELAG grammars, it is desirable that the elements of the same subcategory all have the same inflectional behavior; in this case the `complete` part is made up of only one line per subcategory.

Let us consider for example the following lines from the pronoun description:

```
Pdem  <genre> <nombre>
PpvIl <genre> <nombre> <pers>
PpvPr
```

These lines mean:

- all the demonstrative pronouns (`PRO+Pdem>`) have only a gender and a number;
- clitic pronouns in the nominative (`<PRO+PpvIl>`) are labelled grammatically in person, gender and number;
- the prepositional pronouns (*en*, *y*) do not have any inflectional feature.

All combinations of inflectional features and discriminant subcategories which appear in the dictionaries must be described in the `tagset.def` file, or else the information in the corresponding entries will be discarded by ELAG.

If words of the same subcategory differ by their inflectional profile, it is necessary to write several lines into the `complete` part. The disadvantage of this method of description is that it becomes difficult to make the distinction between such words in an ELAG grammar.

If one considers the description given by the previous example of a `tagset.def` file, certain adjectives of French take a gender and a number, whereas others to not have any inflectional feature. This allows for coding fixed sequences like *de bonne humeur* as adjective, on the basis of their syntactic behavior.

Consider a French dictionary with such sequences as invariable adjectives without inflectional features. The problem is that if one wants to refer exclusively to this type of adjectives in a disambiguation grammar, the `<A>` symbol is not appropriate, since it will recognize all adjectives. To circumvent this difficulty, it is possible to deny an inflectional

attribute by writing the `@` character right before one of the possible values for this attribute. Thus, the `<A:@m@p>` symbol recognizes all the adjectives which have neither a gender nor a number. Using this operator, it is possible to write grammars like those in figure 7.19, which imposes agreement in gender and number between a name and an adjective which suits [2]. This grammar will preserve the correct analysis of sentences like: *Les personnes de bonne humeur m'insupportent*.

Is is however recommended to limit the use of the `@` operator, because it harms the legibility of the grammars. It is preferable to distinguish the labels which accept various inflectional combinations by means of discriminating subcategories defined in the `discr` part.

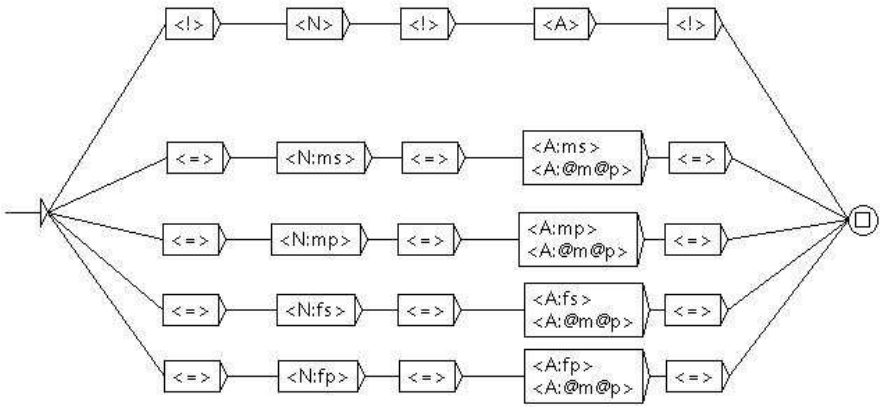


Figure 7.19: ELAG grammar that verifies gender and number agreement

Optional Codes

The optional syntactic and semantic codes are declared in the `cat` part. They can be used in ELAG grammars like other codes. The difference is that these codes do not intervene to decide if a label must be rejected as an invalid one while loading of the text autmaton.

In fact optional codes are independent of other codes, such as for example the attribute of the language level (`z1`, `z2` or `z3`). In the same manner as for inflectional codes, it is possible to deny an inflectional attribute by writing the `!` character right before the name of the attribute. Thus, with our example file, the `<A!gauche:f>` symbol recognizes all adjectives in the feminine which do not have the `gauche` code [3].

All codes which are not declared in the `tagset.def` file are discarded by ELAG. If a dictionary entry contains such a code, ELAG will produce a warning and will withdraw the code from the entry.

[2] This grammar is not completely correct, because it eliminates for example the correct analysis of the sentence: *J'ai reçu des coups de fil de ma mère hallucinants.*

[3] This code indicates that the adjective must appear on the left of the nound to which it refers to, as is the case for *bel*.

Consequently, if two concurrent entries differ in the original text automaton only by undeclared codes, these entries will become indistinguishable by the programs and will thus be unified into only one entry in the resulting automaton.

Thus, the set of labels described in the file `tagset.def` file is compatible with the dictionaries distributed with Unitex, by factorizing words which differ only by undeclared codes, and this independently of the applied grammars.

For example, in the most complete version of the French dictionary, each individual use of a verb is characterized by a reference to the lexicon grammar table which contains it. We have considered until now that this information is more relevant to syntax than to lexical analysis and we thus don't have integrated them into the description of the tagset. They are thus automatically eliminated at the time when the text automaton is loaded, which reduces the rate of ambiguity.

In order to distinguish the effects bound to the tagset from those of the ELAG grammars, it is advised to proceed to a preliminary stage of normalization of the text automaton before applying disambiguation grammars to it. This normalization is carried out by applying to the text automaton a grammar not imposing any constraint, like that of figure 7.20. Note that this grammar is normally present in the Unitex distribution and precompiled in the file norm.rul.

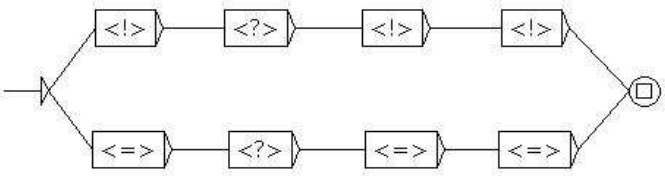


Figure 7.20: ELAG grammar without any constraint

The result of applying such a grammar is that the original is cleaned of all the codes which either are not described in the `tagset.def` file, or do not conform to this description (because of unknown grammatical categories or invalid combinations of inflectional features). By then replacing the text automaton by this normalized automaton, one can be sure that later modifications of the automaton will only be effects of ELAG grammars.

7.3.7 Grammar Optimization

Compilation of ELAG grammars by the `elagcomp` program consists in building an automaton whose language is the set of the sequences of lexical tags (or lexical analyses of a sentence) which are not accepted by the grammars. This task is complex and can take a lot of time. It is however possible to appreciably speed it up by observing certain principles at the time of writing gramars.

Limiting the number of branches in the *then* part

It is recommended to limit the number of *then* parts of a grammar to a minimum. This can reduce considerably the compile time of a grammar. Generally, a grammar having many *then* parts can be rewritten with one or two *then* parts, without a loss of legibility. It is for example the case of the grammar in figure 7.21, which imposes a constraint between a verb and the pronoun which follows it.

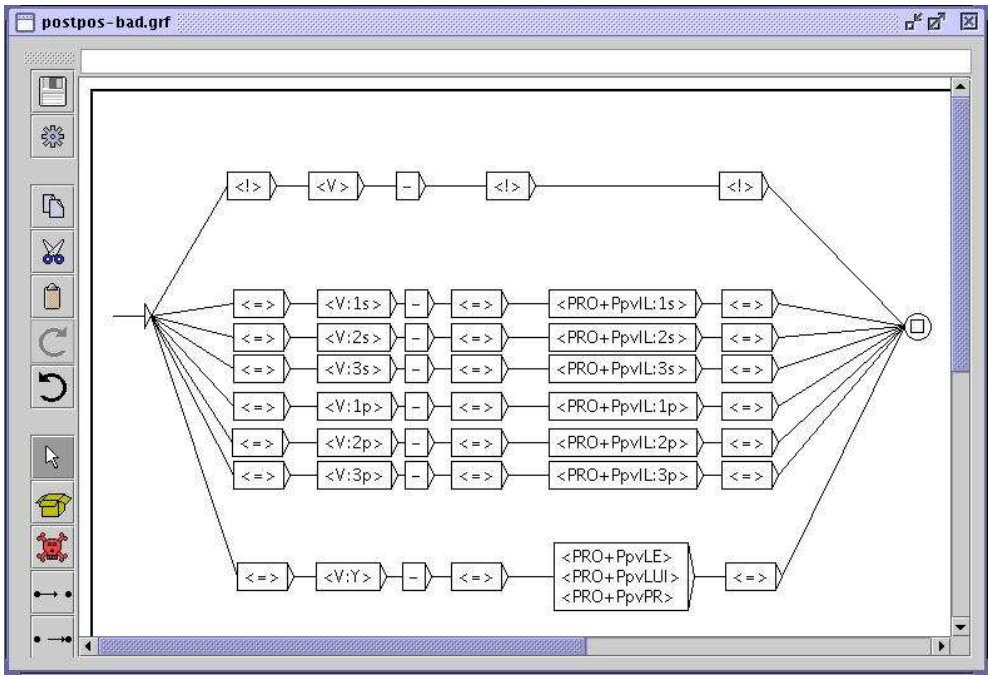


Figure 7.21: ELAG grammar checking verb-pronoun agreement

As one can see in figure 7.22, one can write an equivalent grammar by factorizing all the *then* parts into only one. The two grammars will have exactly the same effect on the text automaton, but the second one will be compiled much more quickly.

Using lexical symbols

It is better to use lemmas only when it is necessary. That is particularly true for some grammatical words, when their subcategories carry almost as much of information as the lemmas themselves. In any case, it is recommended to specify its syntactic, semantic and inflectional features as much as possible. For example, with the dictionaries provided for French, it is preferable to replace symbols like `<je.PRO:1s>`, `<je.PRO+PpvIL:1s>` and `<je.PRO>` with the symbol `<PRO+PpvI1:1s>`. Indeed, all these symbols are identical insofar as they can recognize only the single entry of the dictionary `{je,PRO+PpvIL:1ms:1fs}`. However, as the program does not deduce this information automatically, if all these features are not specified, the program will consider nonexisting labels such as `<je.PRO:3p>`, `<je.PRO+PronQ>` etc. in vain.

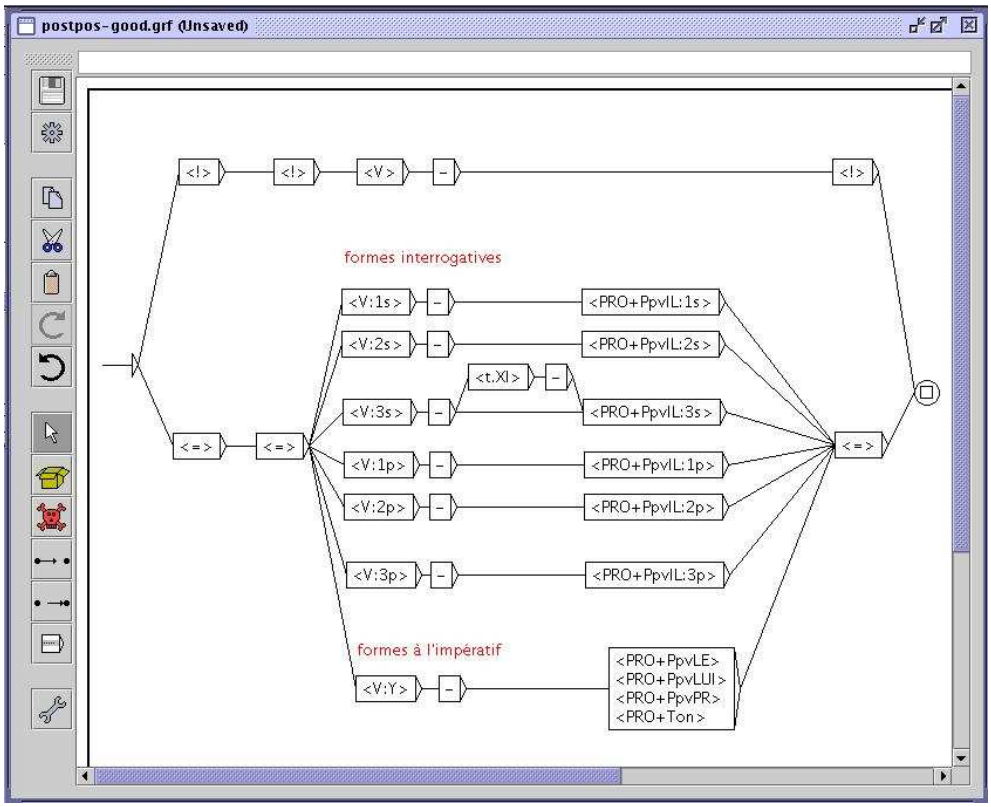


Figure 7.22: Optimized ELAG grammar checking verb-pronoun agreement

7.4 Manipulation of text automata

7.4.1 Displaying sentence automata

As we have seen above, the text automaton is in fact the collection of the sentence automata of a text. This structure can be represented using the format `.fst2`, also used for representing the compiled grammars. This format does not allow the system to directly display the sentence automata. Instead, the system uses the `Fst2Grf` program to convert the sentence automaton into a graph that can be displayed. This program is called automatically when you select a sentence in order to generate the corresponding `.grf` file.

The generated `.grf` files are not interpreted in the same manner as the `.grf` files that represent graphs constructed by the user. In fact, in a normal graph, the lines of a box are separated by the `+` symbol. In the graph of a sentence, each box represents either a lexical unit without a tag or a dictionary entry enclosed by curly brackets. If the box only represents an unlabeled lexical unit, this unit appears alone in the box. If the box represents a dictionary entry, the inflected form is displayed, followed in another line by the canonical form if it is different. The grammatical and inflectional information is displayed below the box as a transducer output.

Figure 7.23 shows the graph obtained for the first sentence of *Ivanhoe*. The words `Ivanhoe`, `Walter` and `Scott` are considered unknown words. The word `by` corresponds to two entries in the dictionary. The word `Sir` corresponds to two dictionary entries as well, but since the canonical form of these entries is `sir`, it is displayed because it differs from the inflected form by a lower case letter.

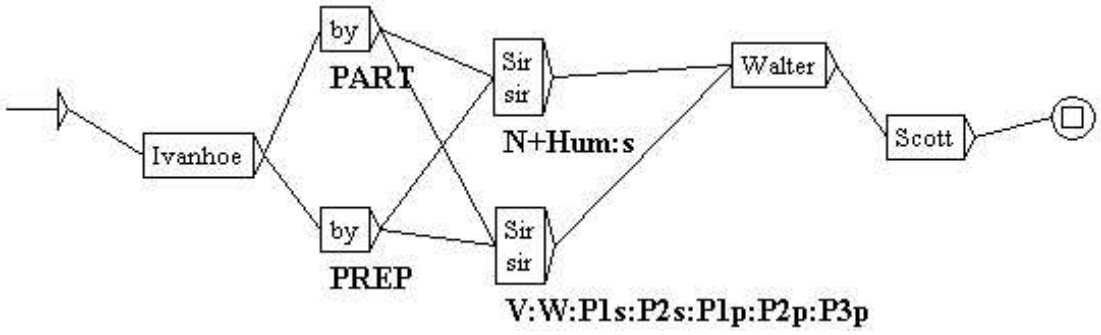


Figure 7.23: Automaton of the first sentence of *Ivanhoe*

7.4.2 Modify the text automaton

It is possible to manually modify the sentence automaton. You can add or erase boxes or transitions. When a graph is modified, it is saved to the text file `sentenceN.grf`, where N represents the number of the sentence.

When you select a sentence, if a modified graph exists for this sentence, this one is displayed. You can then reset the automaton of that sentence by clicking on the botton "Reset Sentence Graph" (cf. figure 7.24).

During the construction of the text automata all the modified sentence graphs in the text file are erased.

NOTE: After you reconstruct the text automaton, you can save your manual modifications. In order to do that, click on the button "Rebuild FST-Text". All sentences that have been modified are then replaced in the text automaton by their modified versions. The new text automaton is then automatically reloaded.

7.4.3 Parameters of presentation

Sentence automata are subject to the same presentation options as the graphs. They use the same colors and fonts as well as the antialiasing effect. In order to configure the appearance of the sentence automata, you modify the general configuration by clicking on "Preferences..." in the "Info" menu. For further details, refer to section 5.3.5.

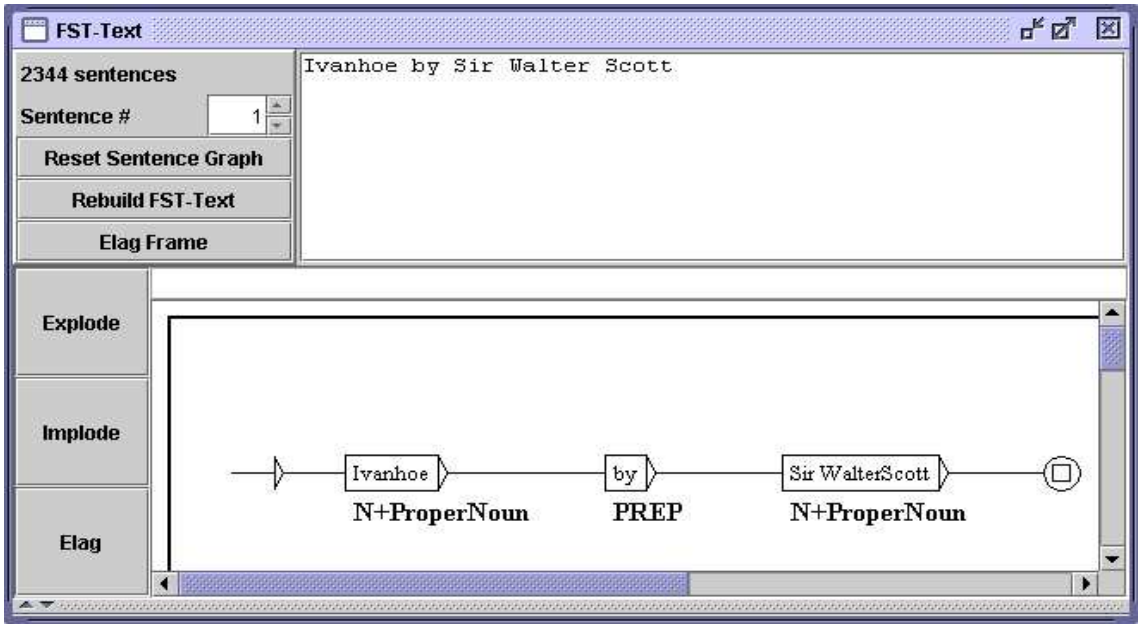


Figure 7.24: Modified sentence automaton

You can also print a sentence automaton by clicking on "Print..." in the "FSGraph" menu or by pressing <Ctrl+P>. Make sure that the printer's page orientation is set to landscape mode. To configure this parameter, click on "Page Setup" in the menu "FSGraph".

7.5 Converting the text automaton into linear text

If the text automaton does not contain any lexical ambiguity, it is possible to build a text file corresponding to the unique path of the automaton. Go into the "Text" menu and click on "Convert FST-Text to Text...". You can set the output text file in the window as shown on Figure 7.25.

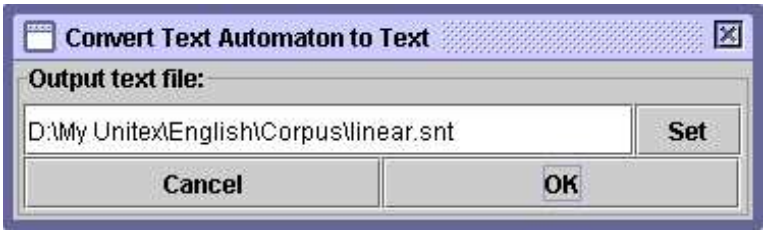


Figure 7.25: Setting output file for linearization of the text automaton

If the automaton is not linear, an error message will give you the number of the first sentence that contain ambiguity. Otherwise, the `Fst2Unambig` program will build the output

file according to the following rules:

- the output file contains one line per sentence;
- every line but the last is ended by `{S};`
- for each box, the program writes its content followed by a space.

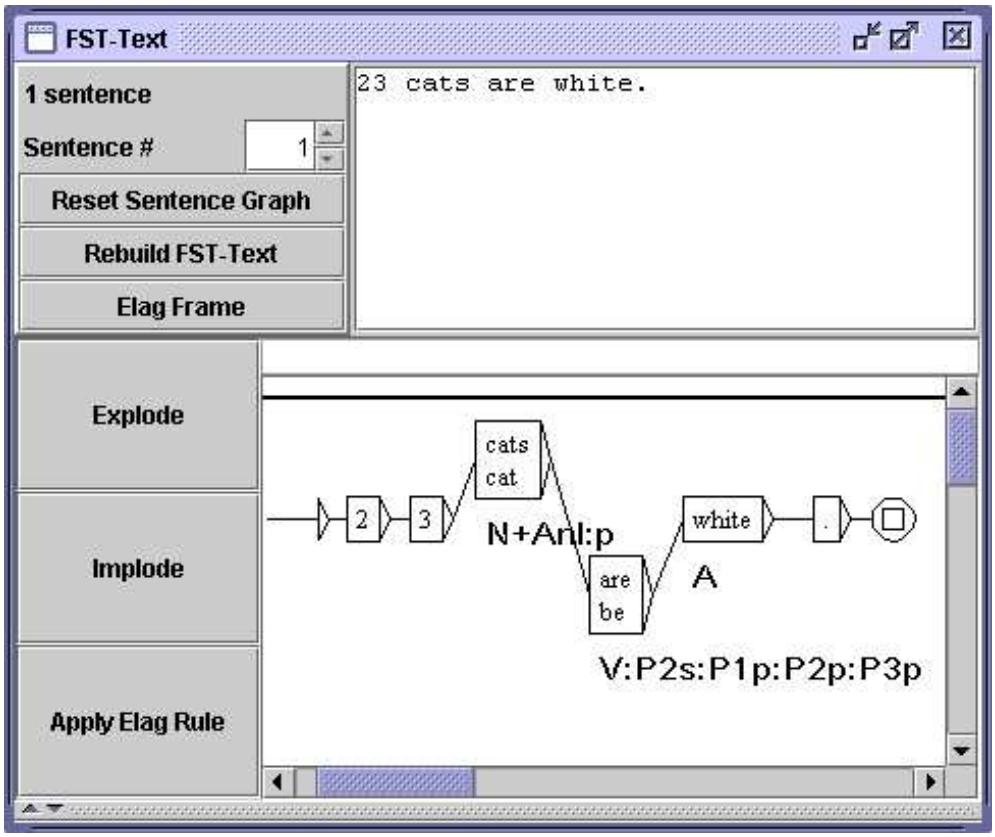


Figure 7.26: Example of a linear text automaton

NOTE: correcting spaces in the output text can only be done manually. If the original text is the one of the text automaton shown on Figure 7.26, the output text will be:

```
2 3 {cats,cat.N+Anl:p} {are,be.V:P2s:P1p:P2p:P3p} {white,white.A} .
```

Chapter 8

Lexicon-grammar

The tables of lexicon-grammar are a compact way of representing syntactical properties of the elements of a language. It is possible to automatically construct local grammars from such tables, due to a mechanism of parameterized graphs.

In the first part of the chapter the formalism of tables is presented. The second part describes parameterized graphs and a mechanism of automatically lexicalizing them with lexicon-grammar tables.

8.1 Lexicon-grammar tables

Lexicon-grammar is a methodology developed by Maurice Gross and the LADL team ([6], [7], [26], [28]) based on the following principle: every verb has an almost unique set of syntactical properties. Due to this fact, these properties need to be systematically described, since it is impossible to predict the exact behavior of a verb. These descriptions are represented by matrices where rows correspond to verbs and columns to syntactical properties. The considered properties are formal properties such as the number and nature of allowed complements of the verb and the different transformations the verb can undergo (passivization, nominalisation, extraposition, etc.). The matrices, or tables, are mostly binary: a + sign occurs at the intersection of a row and a column of a property if the verb has that property, a − sign if not.

This type of description has also been applied to adjectives ([37]), predicative nouns ([21], [22]), adverbs ([27], [39]), as well as frozen expressions, in many languages ([10], [17], [18], [42], [43], [44], [47], [48], [49]).

Figure 8.1 shows an example of a lexicon-grammar table. The table contains verbs that, among other definitional properties, do not admit passivization.

Table32NM.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	N0 = Npc	N0 = N-hum	N0 = Nnr	N0 = V-n	Aux	32NM	N1 V	N1 = Nhum	N1 = N-hum	N1 = le fait Qu P	N1 = V-n	N1 = Dnum Nmes	Ppv = le	N-1 V N0	N0 V Adj	N0 V Dnum V-n	N0 V à N1	N-1 V N0 (<E>+à) N1	N1 = V-n	Exemple
2	-	+	-	-	avoir	accepter	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accepte§vingt personnes
3	-	+	-	-	avoir	accueillir	-	-	+	-	-	-	+	-	-	-	-	-	-	Ce salon§accueille§vingt personnes
4	-	+	-	-	avoir	accuser	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§accuse§80 kilos
5	-	+	-	-	avoir	accuser	-	-	+	+	-	-	-	-	-	-	-	-	-	Max§accuse§ses trente ans
6	-	+	-	-	avoir	admettre	-	-	+	-	-	-	+	-	-	-	-	-	-	On§admet§50 personnes dans cette salle
7	-	+	-	-	avoir	affecter	-	-	+	-	-	-	-	-	-	-	-	-	-	Ces cristaux§affectent§une forme géométrique
8	-	+	-	-	avoir	afficher	-	-	+	-	-	+	-	-	-	-	-	-	-	Les valeurs ont§affiché§un repli
9	-	+	-	-	avoir	aimer	-	-	+	+	-	-	+	-	-	-	-	-	-	La plante§aime§l'eau
10	-	+	-	-	avoir	approcher	+	-	+	-	-	+	+	-	-	-	-	-	-	Cette maison§approche§les deux millions
11	-	+	-	-	avoir	arpenter	-	-	-	-	-	+	+	-	-	+	+	+	-	Ce terrain§arpente§30 arpents
12	-	+	-	-	avoir	atteindre	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§atteint§80 kilos
13	+	+	+	-	avoir	avoir	-	-	+	-	-	+	+	-	-	-	-	-	-	Max§a§(une soeur+une voiture+des sous)
14	-	+	-	-	avoir	avoisiner	-	-	+	-	-	+	+	-	-	-	-	-	-	Ce sac§avoisine§les 20 kg.
15	-	+	-	-	avoir	battre	+	-	+	-	-	-	+	-	-	-	-	-	-	La montre§bat§les secondes
16	-	+	-	-	avoir	cacher	-	-	+	-	-	-	-	-	-	-	-	-	-	Son calme§cache§(son+une grande)angoisse
17	-	+	-	-	avoir	caler	-	-	+	-	-	+	+	-	+	-	-	-	-	Ce bateau§cale§80 cm

c32NM

Figure 8.1: Lexicon-grammar Table 32NM

8.2 Conversion of a table into graphs

8.2.1 Principle of parameterized graphs

The conversion of a table into graphs is carried out by a mechanism involving parameterized graphs. The principle is the following: a graph that describes the possible constructions is constructed manually. That graphs refers to the columns of the table in the form of parameters or variables. Afterwards, for each line of the table a copy of this graph is constructed where the variables are replaced with the contents of the cell at the intersection of line and the column that corresponds to the variable. If a cell of the table contains the + sign, the corresponding variable is replaced by <E>. If the cell contains the – sign, the box containing the corresponding variable is removed, interrupting the paths through that box. In all other cases the variable is replaced by the contents of the cell.

8.2.2 Format of the table

The lexicon-grammar tables are usually encoded with the aid of a spreadsheet like OpenOffice.org Calc ([41]). To make them usable with Unitex, the tables have to be encoded in Unicode text format in accordance with the following convention: the columns need to be separated by a tab and the lines by a newline.

In order to convert a table with OpenOffice.org Calc, save it in text format (`.csv` extension). You can then parameterize the output format with a window as shown on Figure 8.2. Choose "Unicode", select tabulation as column separator and do not set any text delimiter.

Figure 8.2: Saving a table with OpenOffice.org Calc

During the generation of the graphs, Unitex skips the first line, considering it contains the headings of the columns. It is therefore necessary to ensure that the headings of the columns occupy exactly one line. If there is no line for the heading, the first line of a table will be ignored anyway, and if there are multiple heading lines, from the second line on they will be interpreted as lines of the table.

8.2.3 Parameterized graphs

Parameterized graphs are graphs with variables referring to the columns of a table of the lexicon-grammar. This mechanism is usually used with syntactical graphs, but nothing prevents the construction of parameterized graphs for inflection, preprocessing, or for normalization.

The variables that refer to columns are formed with the `@` symbol followed by the name of the column in capital letters (the columns are named starting with `A`).

Example: `@C` refers to the third column of the table.

Whenever a variable takes the value of a + or − sign, the − sign corresponds to the removal of a path through that variable. It is possible to swap the meaning of these signs by typing an exclamation mark in front of the `@` symbol. In that case, the path is removed when there is a + sign and keeped where there is a − one. In all other cases, the variable is replaced by the content of the table cell.

The special variable `@%` is replaced by the number of the line in the table. The fact that its value is different for each line allows for its use as a simple characterization of a line. That variable is not affected by an exclamation point to the left of it.

Figure 8.3 shows an example of a parameterized graph designed to be applied to the lexicon-grammar table 31H presented in figure 8.4.

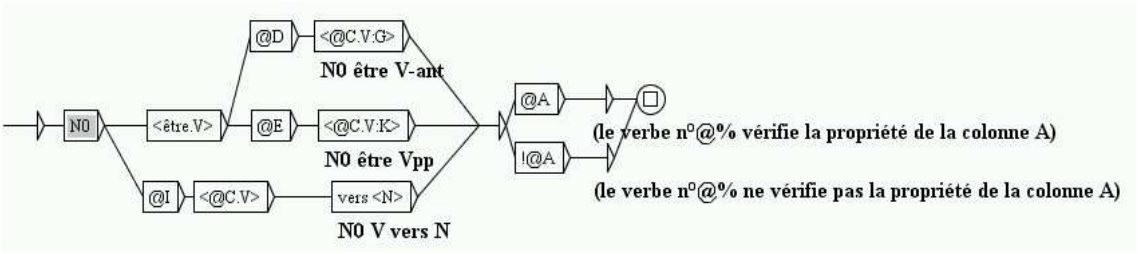


Figure 8.3: Example of parameterized graph

Table31H.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	N0 = V-n	Aux	31H	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0pc	Nhum V sur ce point	N0 V vers N	il V N0 W	idée Loc esprit	Nhum Loc Nabs	N0 = N-hum	N0 = V-n	Exemple
2	-	avoir	abandonner	-	-	-	-	-	-	-	-	-	-	-	Paul a§abandonné§
3	-	avoir	abuser	-	-	-	-	+	-	-	-	-	-	-	Max§abuse§
4	-	avoir	acquiescer	-	-	-	+	+	-	-	-	-	-	-	Max a§acquiescé§(E+de la tête)
5	-	avoir	adouber	-	-	-	-	-	-	-	-	-	-	-	Paul§adoube§|échecs|
6	-	avoir	agioter	-	-	-	-	-	-	+	-	-	-	-	Max§agiote§sur les changes
7	-	avoir	agoniser	+	-	-	-	-	-	+	-	-	+	-	Max§agonise§
8	-	avoir	archaïser	+	-	-	-	+	-	-	-	-	-	-	Cet auteur§archaïse§volontiers
9	-	avoir	arquer	-	-	-	+	-	+	-	-	-	-	-	Max a§arqué§toute la journée
10	-	être	arriver	-	+	-	-	-	-	+	-	+	-	-	Max est§arrivé§
11	-	avoir	atermoyer	-	-	-	-	+	-	+	-	+	-	-	Max§atermoie§
12	+	avoir	badauder	-	-	-	-	-	+	-	+	-	-	badaud	Max§badaude§

c31H

Figure 8.4: Lexicon-grammar table 31H

8.2.4 Automatic generation of graphs

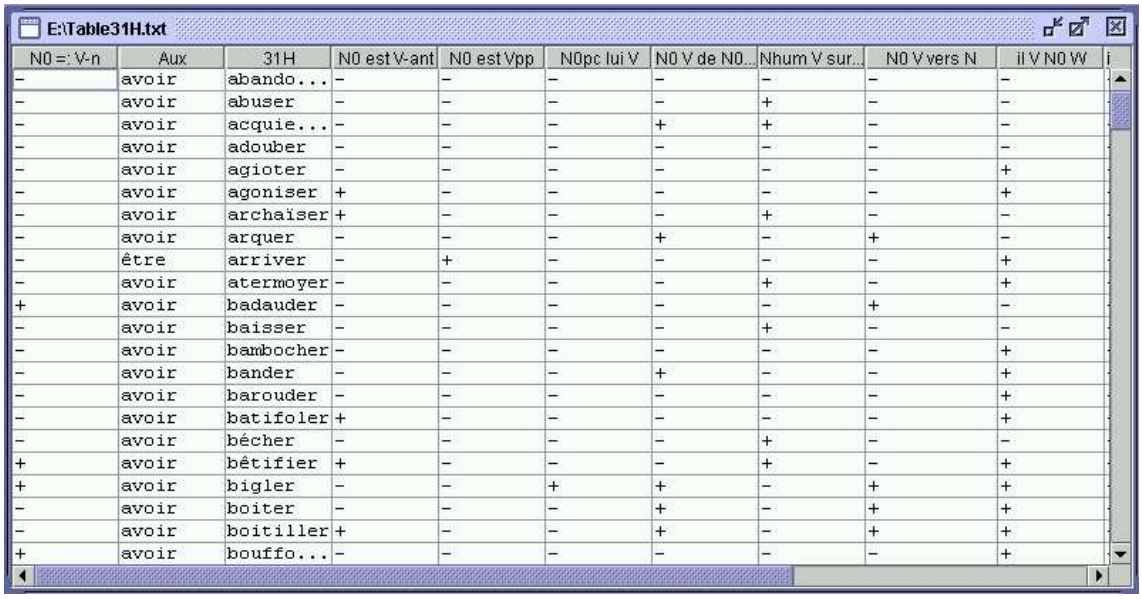
In order to be able to generate graphs from a parameterized graph and a table, first of all the table must be opened by clicking on "Open..." in the menu "Lexicon-Grammar" (see figure 8.5). The table must be in Unicode text format.

The selected table is then displayed in a window (see figure figure 8.6). If it that does appear on your screen, it may be hidden by other Unitex windows.

To automatically generate graphs from a parameterized graph, click on "Compile to GRF..." in the menu "Lexicon-Grammar". The window in figure 8.7 shows this.

In the frame "Reference Graph (in GRF format)", indicate the name of the parameterized graph to be used. In the frame "Resulting GRF grammar", indicate the name of the main

Figure 8.5: Menu "Lexicon-Grammar"


E:\Table31H.txt

N0 =: V-n	Aux	31H	N0 est V-ant	N0 est Vpp	N0pc lui V	N0 V de N0...	Nhum V sur...	N0 V vers N	il V N0 W
-	avoir	abando...	-	-	-	-	-	-	-
-	avoir	abuser	-	-	-	-	+	-	-
-	avoir	acquie...	-	-	-	+	+	-	-
-	avoir	adouber	-	-	-	-	-	-	-
-	avoir	agioter	-	-	-	-	-	-	+
-	avoir	agoniser	+	-	-	-	-	-	+
-	avoir	archaïser	+	-	-	-	+	-	-
-	avoir	arquer	-	-	-	+	-	+	-
-	être	arriver	-	+	-	-	-	-	+
-	avoir	atermoyer	-	-	-	-	+	-	+
+	avoir	badauder	-	-	-	-	-	+	-
-	avoir	baisser	-	-	-	-	+	-	-
-	avoir	bambocher	-	-	-	-	-	-	+
-	avoir	bander	-	-	-	+	-	-	+
-	avoir	barouder	-	-	-	-	-	-	+
-	avoir	batifoler	+	-	-	-	-	-	+
-	avoir	bécher	-	-	-	-	+	-	-
+	avoir	bétifier	+	-	-	-	+	-	+
+	avoir	bigler	-	-	+	+	-	+	+
-	avoir	boiter	-	-	-	+	-	+	+
-	avoir	boitiller	+	-	-	+	-	+	+
+	avoir	bouffo...	-	-	-	-	-	-	+

Figure 8.6: Displaying a table

graph that will be generated. This main graph is a graph that invokes all the graphs that are going to be generated. When launching a search in a text with that graph, all the generated graphs are simultaneously applied.

The "Name of produced subgraphs" frame is used to set the name of each graph that will be generated. Enter a name containing `@%`, because for each line of the table, `@%` will be replaced the line number, which guarantees that each graph name will be unique. For example, if the main graph is called "`TestGraph.grf`" and if subgraphs are called "`TestGraph_@%.grf`", the graph generated from the 16th line of the line will be named "`TestGraph_0016.grf`".

Figures 8.8 and 8.9 show two graphs generated by applying the parameterized graph of figure 8.3 at table 31H.

Figure 8.10 shows the resulting main graph.

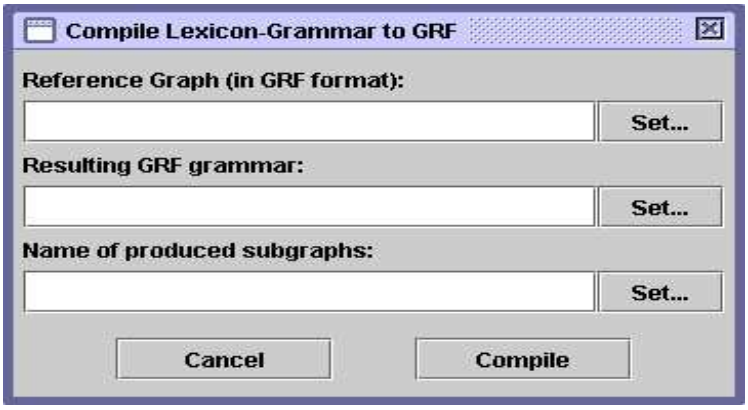


Figure 8.7: Configuration of the automatic generation of graphs

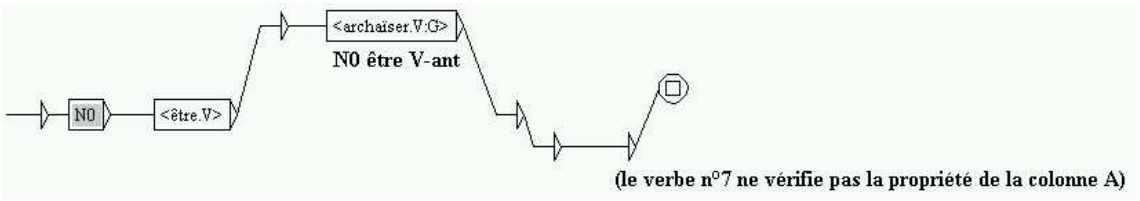


Figure 8.8: Graph generated for the verb `archaïser`

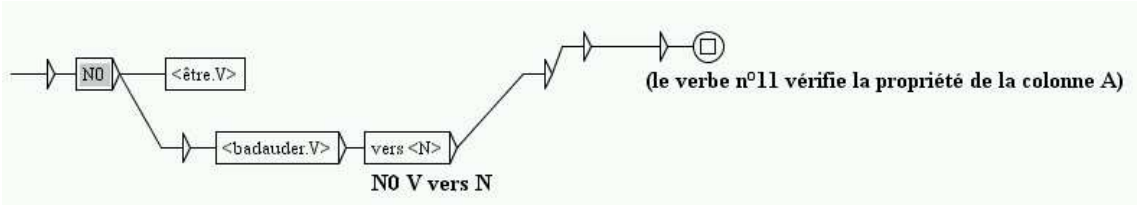


Figure 8.9: Graph generated for the verb `badauder`

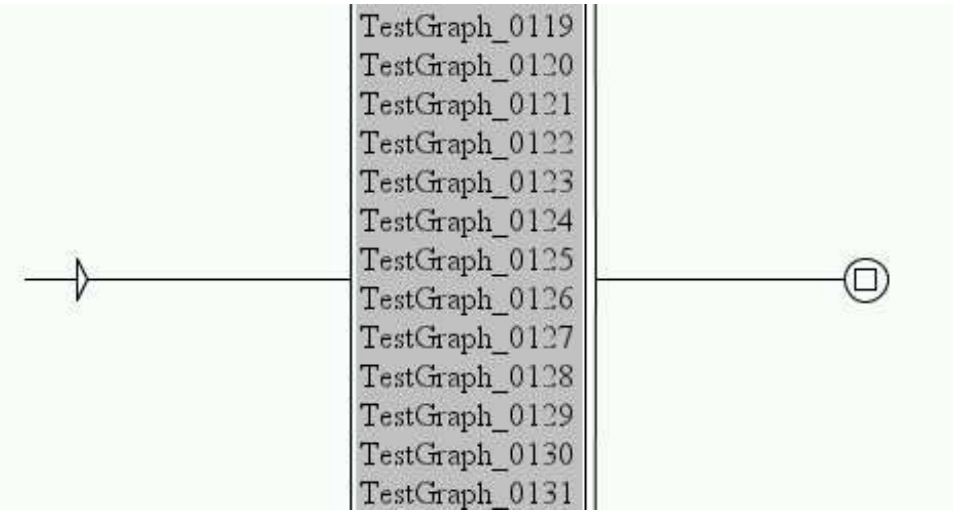


Figure 8.10: Main graph referring to all the generated graphs

Chapter 9

Use of external programs

This chapter presents the use of the different programs of which Unitex is composed. These programs, which can be found in the `Unitex/App` folder, are automatically called by the interface. It is possible to see the commands that have been executed by clicking on the menu "Info" on the "Console". It is also possible to see the options of the different programs in the selection of the "Help on commands" sub-menu of the "Info" menu.

WARNING: multiple programs use the text directory (`my_text_snt`). This directory is created by the graphical interface after the normalization of the text. If you work with the command line, you have to create the directory manually before the execution of the program `Normalize`.

WARNING (2): whenever a parameter contains spaces, it needs to be enclosed in quotation marks so it will not be considered as multiple parameters.

9.1 CheckDic

```
CheckDic dictionnaire type
```

This program carries out the verification of the format of a dictionary of DELAS or DELAF type. The parameter `dictionnaire` corresponds to the name of the dictionary that is to be verified. The parameter `type` can take the value `DELAS` or `DELAF` depending on the format of the dictionary to be verified.

The program checks the syntax of the lines of the dictionary. It also creates a list of all characters occurring in the inflected and canonical forms of words in the text, the list of grammatical codes and syntax, as well as the list of inflection codes used. The results of the verification are stored in a file called `CHECK_DIC.TXT`.

9.2 Compress

```
Compress dictionary [-flip]
```

This program takes a DELAF dictionary as a parameter and compresses it. The compression of a dictionary `dico.dic` produces two files:

- `dico.bin`: a binary file containing the minimum automaton of the inflected forms of the dictionary;
- `dico.inf`: a text file containing the compressed forms required for the reconstruction of the dictionary lines from the inflected forms contained in the automaton.

For more details on the format of these files, see chapter 10. The optional parameter `-flip` indicates that the inflected and canonical forms should be swapped in the compressed dictionary. This option is used to construct an inverse dictionary which is necessary for the program `Reconstrucao`.

9.3 Concord

```
Concord index font fontsize left right order mode alph [-thai]
```

This program takes a concordance index file produced by the program `Locate` and produces a concordance. It is also possible to produce a modified text version taking into account the transducer outputs associated to the occurrences. Here is the description of the parameters:

- `index`: name of the concordance index file. It is necessary to indicate the entire file path, since Unitex uses it to determine for which text the concordance is to be constructed.
- `font`: name of the typeface if the concordance is in HTML format. This value is ignored if the concordance is not in HTML format.
- `fontsize`: size of the typeface if the concordance is in HTML format. This value has to be between 1 and 7. Like parameter `font`, it is also ignored, if the concordance is not in HTML format.
- `left`: number of characters on the left of the occurrences. In Thai mode, this means the number of non-diacritic characters.
- `right`: number of characters (non-diacritic ones in Thai mode) on the right of the occurrences. If the occurrence is shorter than this value, the concordance line is completed up to `right`. If the occurrence is longer than the length defined by `right`, it is nevertheless saved as whole.
- `order`: indicates the mode to be used to sort the lines of the concordance. The possible values are:
  - `TO`: order in which the occurrences appear in the text;
  - `LC`: left context for primary sort, then occurrence for secondart sort;

  - LR: left context, then right context;
  - CL: occurrence, then left context;
  - CR: occurrence, then right context;
  - RL: right context, then left context;
  - RC: left context, then occurrence.
  - NULL: does not specify any sorting mode. This option should be used if the text is to be modified instead of constructing a concordance.

  For details on the sorting modes, see section 4.8.2.

- mode: indicates in which format the concordance is to be produced. The four possible formats are:
  - html: produces a concordance in HTML format encoded in UTF-8;
  - text: produces a concordance in Unicode text format;
  - glossanet: produces a concordance for GlossaNet in HTML format. The HTML file is encoded in UTF-8;
  - name_of_file: indicates to the program that it is supposed to produce a modified version of the text and save it in a file named name_of_file (see section 6.7.3).
- alph: alphabet file used for sorting. The value NULL indicates the absence of an alphabet file.
- -thai: this parameter is optional. It indicates to the program that it is processing a Thai text. This option is necessary to ensure the proper functioning of the program in Thai.

The result of the application of this program is a file called concord.txt if the concordance was constructed in text mode, a file called concord.html if the mode was html or glossanet, and a text file with the name defined by the user of the program if the program has constructed a modified version of the text.

In html mode, the occurrence is coded as a hypertext link. The reference associated to this link is of the form <a href="X Y Z">. X et Y represent the beginning and ending positions of the occurrence of the characters in the file nom_du_texte.snt. Z represents the number of the sentence in which the occurrence was found.

9.4 ConcorDiff

```
ConcorDiff concor1 concor2 out font size
```

This program takes 2 concordance files and produces an HTML page that shows their differences (see section 6.7.5, page 107). The parameters are:

- `concor1` and `concor2` : concordance index files (`.ind`). These file names must be absolute, because Unitex uses these names to deduce on which text there were computed;
- `out`: output HTML page;
- `font`: font to use in output HTML page;
- `size`: font size to use in output HTML page.

9.5 Convert

```
Convert src [dest] mode text_1 [text_2 text_3 ...]
```

With this program you can transcode text files. `src` indicates the input encoding. The optional `dest` parameter indicates the output encoding. The default output value is `LITTLE-ENDIAN`. These parameters can take the following values:

`FRENCH`
`ENGLISH`
`GREEK`
`THAI`
`CZECH`
`GERMAN`
`SPANISH`
`PORTUGUESE`
`ITALIAN`
`NORWEGIAN`
`LATIN` (default latin code page)
`windows-1252`: Microsoft Windows 1252 - Latin I (Western Europe & USA)
`windows-1250`: Microsoft Windows 1250 - Central Europe
`windows-1257`: Microsoft Windows 1257 - Baltic
`windows-1251`: Microsoft Windows 1251 - Cyrillic
`windows-1254`: Microsoft Windows 1254 - Turkish
`windows-1258`: Microsoft Windows 1258 - Viet Nam
`iso-8859-1` : ISO 8859-1 - Latin 1 (Europe de l'ouest & USA)
`iso-8859-15` : ISO 8859-15 - Latin 9 (Western Europe & USA)
`iso-8859-2` : ISO 8859-2 - Latin 2 (Eastern and Central Europe)
`iso-8859-3` : ISO 8859-3 - Latin 3 (Southern Europe)
`iso-8859-4` : ISO 8859-4 - Latin 4 (Northern Europe)
`iso-8859-5` : ISO 8859-5 - Cyrillic
`iso-8859-7` : ISO 8859-7 - Greek
`iso-8859-9` : ISO 8859-9 - Latin 5 (Turkish)
`iso-8859-10` : ISO 8859-10 - Latin 6 (Nordic)
`next-step` : NextStep code page
`LITTLE-ENDIAN`

BIG-ENDIAN

NOTE: there is an additional mode for dest with the value UTF-8 that indicates that the program must convert files into UTF-8 files.

mode specifies how to handle name of input and output files. The following values are possible:
-r : input files are overwritten
-ps=PFX : input files are renamed with the PFX prefix (toto.txt ⇒ PFXtoto.txt)
-pd=PFX : ouput files are renamed with the PFX prefix
-ss=SFX : input files are named with the SFX suffix (toto.txt ⇒ totoSFX.txt)
-sd=SFX : ouput files are named with the SFX suffix

text_i stand for the files to be converted.

9.6 Dico

```
Dico text alphabet dic_1 [dic_2 ...]
```

This program applies dictionaries to a text. The text must have been cut up into lexical units by the program Tokenize.

text represents the complete file path, without omitting the extension .snt. dic_i represents the path and name of a dictionary. The dictionary must be a .bin dictionary (obtained with the Compress program) or a dictionary graph in the .fst2 format (see section 3.6, page 44). It is possible to give priorities to the dictionaries. For details see section 3.6.1.

The program Dico produces the following four files, and saves them in the directory of the text:

- dlf: dictionary of simple words in the text;
- dlc: dictionary of composed words in the text;
- err: list of unknown words in the text;
- stat_dic.n: file containing the number of simple words, the number of composed words, and the number of unknown words in the text.

NOTE: the files dlf, dlc and err are not sorted. Use the program SortTXT to sort them.

9.7 Elag

```
Elag txtauto -l lang -g rules -o output [-d dir]
```

This program takes a text automaton `txtauto` and applies to it ambiguity removal rules. Parameters are:

- `txtauto`: a `.fst2` text automaton;
- `lang` : ELAG configuration file for the language of the text;
- `rules` : rule file compiled in the `.rul` format;
- `output` : the output text automaton;
- `dir` : this optional parameter indicates the directory where ELAG rules are located.

9.8 ElagComp

```
ElagComp [-r ruleslist|-g grammar] -l lang [-o output] [-d rulesdir]
```

This program compiles the ELAG grammar named `grammar`, or all the grammars specified in the `ruleslist` file. The result is stored in the `output` file that will be used by the `Elag` program.

- `ruleslist` : file listing ELAG grammars;
- `lang` : ELAG configuration file for the language of the grammar(s);
- `output` : this optional parameter specifies the output file. By default, the output file name is the same as `ruleslist`, except for the extension that is `.rul`;
- `rulesdir` : this optional parameter indicates the directory where ELAG rules are located.

9.9 Evamb

```
Evamb [-imp|-exp] [-o] fstname [-n sentenceno]
```

This program computes an average lexical ambiguity rate on the text automaton `fstname`, or just on the sentence which number is specified by `sentenceno`. If `-imp` is used, the computation is done on the *compact* form of the automaton, in which inflectional ambiguity is ignored. If `-exp` is used, all instances of lexical ambiguity are taken into account (*developed* form of the automaton). Thus, the entry `aimable,.A:ms:fs` will be counted once with `-imp` and twice with `-exp`. The results of the computation are displayed on the standard output. The text automaton is not modified.

9.10 ExploseFst2

```
ExploseFst2 txtauto -o out
```

This program computes and stores in `out` the *developed* form of the text automaton `txtauto`.

9.11 Extract

```
Extract yes/no text concordance result
```

This program takes a text and a concordance as parameters. If the first parameter is `yes`, the program extracts from the text all sentences that contain at least one occurrence from the concordance. If the parameter is `no`, the program extracts all sentences that do not contain any occurrences from the concordance. The parameter `text` represents the complete path of the text file, without omitting the extension .snt. The parameter `concordance` represents the complete path of the concordance file, without omitting the extension `.ind`. The parameter `result` represents the name of the file in which the extracted sentences are to be saved.

The result file is a text file that contains all extracted sentences, one sentence per line.

9.12 Flatten

```
Flatten fst2 type [depth]
```

This program takes an ordinary grammar as its parameter, and tries to transform it into a final state transducer. The parameter `fst2` indicates the grammar to be transformed. The parameter `type` indicates which kind of grammar the result grammar should be. If this parameter is `FST`, the grammar is "unfolded" to maximum depth and is truncated if there are calls to sub-graphs. Truncated calls are replaced by void transitions. The result is a grammar in `.fst2` format that does only contain a single finite state transducer.

If the parameter is `RTN`, the calls to sub-graphs that could remain after the transformation are left as they are. The result is therefore a finite state transducer in the favorable case, and an optimized grammar strictly equivalent to the original grammar if not. The optional parameter `depth` indicates the maximum depth to which graph calls should be unfolded. The default value is 10.

9.13 Fst2Grf

```
Fst2Grf text_automaton sentence [output] [-f=font]
```

This program extracts a sentence automaton in `.grf` format from the automaton of a text. The parameter `text_automaton` represents the complete path of the file of the text

automaton from which a sentence is to be extracted. This file is called `text.fst2` and is stored in the directory of the text. The parameter `sentence` indicates the number of the sentence to be extracted.

The program produces the following two files and saves them in the directory of the text:

- `cursentence.grf`: graph representing the automaton of the sentence
- `cursentence.txt`: text file containing the sentence.

If you specify the optional `output` parameter, the output files will be named `output.grf` and `output.txt`, instead of `cursentence.grf` and `cursentence.txt`. With the optional parameter `-f=font`, you can specify the font to be used in the output graph. The default font is *Times new Roman*.

9.14 Fst2List

```
Fst2List [-o out][-p s/f/d][-[a/t] s/m][-f s/a][-s "L,[R]"]
            [-s0 "Str"][-v][-rx "L,[R]"] [-l line#] [-i subname]*
            [-c SS=0xxxx]* fname
```

This program takes a `.fst2` file and lists the sequences recognized by this grammar. The parameters are:

- `fname` : grammar name, including `.fst2`;
- `-o out` : specifies the output file, `lst.txt` by default;
- `-[a/t] s/m` : indicates if the program must take into account (`t`) or not (`a`) the outputs of the grammars if any. `s` indicates that there is only one initial state, whereas `m` indicates that there are several ones (this mode is useful in Korean). The default value is `-a s`;
- `-l line#` : maximum number of lines to be printed in the output file;
- `-i subname` : indicates that the recursive exploration must end when the program enters in graph `subname`. This parameter can be used several times in order to specify several stop graphs;
- `-p s/f/d` : `s` displays paths graph by graph; `f` (default) displays global paths; `d` displays global paths with information on nested graph calls;
- `-c SS=0xXXXX`: replaces symbol `SS` when it appears between angle brackets by the Unicode character whose hexadecimal number is `0xXXXX`;
- `-s "L[,R]"` : specifies the left (`L`) and right (`R`) delimiters that will enclose items. By default, no delimiters are specified;

- `-s0 "Str"` : if the program must take outputs into account, this parameter specifies the sequence `Str` that will be inserted between input and output. By default, there is no separator;
- `-f a/s` : if the program must take outputs into account, this parameter specifies the format of the lines that will be generated: `in0 in1 out0 out1` (`s`) or `in0 out0 in1 out1` (`a`). The default value is `s`;
- `-v` : prints information during the process (verbose mode);
- `-rx "L,[R]"`: specifies how cycles must be displayed. `L` and `R` are delimiters. If we consider the graph shown on Figure 9.1, here are the results for `L="[`" and `R="]*"`:

```
il fait [très très]*
il fait très beau
```

Figure 9.1: Graph with a cycle

9.15 Fst2Txt

```
Fst2Txt text fst2 alph mode [-char_by_char|-char_by_char_with_space]
```

This program applies a transducer to a text at the preprocessing stage, when the text has not been cut into lexical units yet. The parameters of the program are the following:

- `text`: the text file to be modified, with extension `.snt`;
- `fst2`: the transducer to be applied;
- `alph`: the alphabet file of the language of the text;
- `mode`: the application mode of the transducer. The two possible modes are `-merge` and `-replace`;
- `-char_by_char`: with this optional parameter, the transducer applies in "character by character" mode. This option is used for texts in Asian languages;
- `-char_by_char_with_space`: with this optional parameter, the transducer applies in "character by character" mode. It also allows to match sequences even if they begin with a space.

This program modifies the text file given as a parameter.

9.16 Fst2Unambig

```
Fst2Unambig fst2 output
```

This programs takes a text automaton `fst2` and produces an equivalent text file `output` if the automaton is linear (with no ambiguity). See section 7.5, page 134.

9.17 Grf2Fst2

```
Grf2Fst2 graph [y/n] [alph] [-d repository]
```

This program compiles a grammar into a `.fst2` file (for more details see section 6.2). The parameter `graph` denotes the complete path of the main graph of the grammar, without omitting the extension `.grf`.

The `y/n` parameter is optional. It indicates to the program whether the grammar needs to be checked for errors or not. By default, the program carries out this error check.

The `alph` parameter specifies the alphabet file to be used for tokenizing the content of the grammar boxes into lexical units. If the value given to this parameter is `char_by_char`, then the tokenization will be done character by character. If it is omitted, lexical units will be sequences of any Unicode letters.

The optional `-d repository` parameter specifies the repository directory to use (see section 5.2.3, page 70).

The result is a file with the same name as the graph passed to the program as a parameter, but with extension `.fst2`. This file is saved in the same folder as `graph`.

9.18 ImploseFst2

```
ImploseFst2 txtauto -o out
```

This program computes and stores in `out` the *compact* form of the text automaton `txtauto`.

9.19 Inflect

```
Inflect delas result dir [-a] [-k]
```

This program carries out the automatic inflection of a DELAS dictionary. The parameter `delas` indicates the name of the dictionary to be inflected. The parameter `result` indicates the name of the dictionary to be generated. The parameter `dir` indicates the complete file path of the directory which contains the inflection transducers that the `delas` dictionary refers to. The optional `-a` parameter indicates that the `:` character will be inserted when the sequence produced by an inflectional transducer does not start with `:`. The optional `-k` parameter indicates that grammatical codes in the inflected dictionary will be identical to inflectional transducer names (`N32` will not be turned into `N`).

The result of the inflection is a DELAF dictionary saved under the name indicated by the parameter `result`.

9.20 Locate

```
Locate text fst2 alphabet s/l/a i/m/r n [dir] [-thai] [-space]
```

This program applies a grammar to a text and constructs an index of the occurrences found. The parameters are as follows:

- `text`: complete path of the text file, without omitting the `.snt` extension;
- `fst2`: complete path of the grammar, without omitting the `.fst2` extension;
- `alphabet`: complete path of the alphabet file;
- `s/l/a`: parameter indicating whether the search should be carried out in *shortest matches* (`s`), *longest matches* (`l`) ou *all matches* (`a`) mode;
- `i/m/r`: parameter indicating the application mode of the transductions: mode MERGE (`m`) or mode REPLACE. `i` indicates that the program should not take into account transducer outputs;
- `n`: parameter indicating the maximum number of occurrences to be searched for; The value `all` indicates that all occurrences need to be extracted;
- `dir`: optional directory that will replace the default `..._snt` one. It must end with a file separator (`\` or `/`);
- `-thai`: optional parameter necessary for searching a Thai text;
- `-space`: optional parameter indicating that the search will start at any position in the text, even before a space. This parameter should only be used to carry out morphological searches.

This program saves the references to the found occurrences in a file called `concord.ind`. The number of occurrences, the number of units belonging to those occurrences, as well as the percentage of recognized units within the text are saved in a file called `concord.n`. These two files are stored in the directory of the text.

9.21 MergeTextAutomaton

```
MergeTextAutomaton automaton
```

This program reconstructs text automaton `automaton` taking into account the manual modifications. If the program finds a file `sentenceN.grf` in the same directory as `automaton`, it replaces the automaton of sentence `N` with the one represented by `sentenceN.grf`. The `automaton` file is replaced by the new text automaton. The old text automaton is backed up in a file called `text.fst2.bck`.

9.22 Normalize

```
Normalize txt [-no_CR]
```

This program carries out a normalization of text separators. The separators are space, tab, and newline. Every sequence of separators that contains at least one newline is replaced by a unique newline. All other sequences of separators are replaced by a single space.

This program also verifies the syntax of the lexical tags in the text. All sequences in curly brackets is either the sentence delimiter `{S}`, or a valid entry in the DELAF format (`{aujourd'hui,.ADV}`). If the program finds curly brackets that are used differently, it gives a warning and replaces them by square brackets (`[` and `]`). Parameter `txt` represents the complete path of the text file. The program creates a modified version of the text that is saved in a file with extension `.snt`.

The optional parameter `-no_CR` replaces any separator sequence by a single space.

9.23 PolyLex

```
PolyLex lang alph dic list out [info]
```

This program takes a file with unknown words `list` and tries to analyse each of the words as a compound obtained by concatenating simple words. The words that have at least one analysis are removed from the file of unknown words and the dictionary lines that correspond to the analysis are appended to file `out`. Parameter `lang` determines the language. The possible values are `GERMAN`, `NORWEGIAN` and `RUSSIAN`. Parameter `alph` represents the alphabet file to be used. Parameter `dic` designates which dictionary will be consulted for the analysis. Parameter `out` designates the file in which the produced dictionary lines are to be printed; if that file already exists, the produced lines are appended at the end of the file. Optional parameter `info` designates a text file in which the information about the analysis has been produced.

9.24 Reconstrucao

```
Reconstrucao alph concord dic reverse_dic pro nasalpro res
```

This program generates a normalization grammar designed to be applied before the construction of an automaton for a Portuguese text. Parameter `alph` designates the alphabet file to be used. The `concord` file represents a concordance which has to be produced by applying in MERGE mode to the considered text a grammar that extracts all forms to be normalized. This grammar is called `V-Pro-Suf`, and is stored in the `/Portuguese/Graphs/Normalization` directory. The `dic` parameter designates which dictionary will be used to find the canonical forms that are associated to the roots of the verbs. `reverse_dic` designates the inverse dictionary to use to find forms in the future and conditional given their canonical forms. These two dictionaries have to be in `.bin` format, and `reverse_dic` has to be obtained by compressing the dictionary of verbs in the future and conditional with the parameter `-flip`

(see section 9.2). Parameter `pro` designates the grammar for rewriting pronouns. Parameter `nasalpro` designates the grammar for rewriting nasal pronouns. `res` designates the file `.grf` into which the normalization rules are to produce.

9.25 Reg2Grf

```
Reg2Grf fic
```

This program constructs a `.grf` file corresponding to the regular expression written in file `fic`. The parameter `fic` represents the complete path to the file containing the regular expression. This file needs to be a Unicode text file. The program takes into account all characters up to the first newline. The result file is called `regexp.grf` and is saved in the same directory as `fic`.

9.26 SortTxt

```
SortTxt text [OPTIONS]
```

This program carries out a lexicographical sorting of the lines of file `text`. `text` represents the complete path of the file to be sorted. The possible options are:

- `-y`: delete repeated lines;
- `-n`: conserve repeated lines;
- `-r`: sort in descending order;
- `-o fic`: sort using the alphabet of the order defined by file `fic`. If this parameter is missing, the sorting is done according to the order of Unicode characters;
- `-l fic`: backup the number of lines of the result file in file `fic`;
- `-thai`: option for sorting a Thai text.

The sort operation modifies file `text`. By default, the sorting is performed in the order of Unicode characters, removing repeated lines.

9.27 Table2Grf

```
Table2Grf table grf result.grf [pattern]
```

This program automatically generates graphs from a lexicon-grammar `table` and template graph `grf`. The name of the produced main graph of the grammar is `result.grf`.

If the `pattern` parameter if specified, all the produced subgraphs will be named according to this pattern. In order to have unambiguous names, we recommend to include

`@%` in the parameter (remind that `@%` will be replaced by the line number of the entry in the table). For instance, if you set the pattern parameter to '`subgraph-@%.grf`', subgraphs names will be such as '`subgraph-0013.grf`'. If the pattern parameter is omitted, subgraphs name like '`result_0013.grf`', where '`result.grf`' designates the result main graph.

9.28 TagsetNormFst2

```
TagsetNormFst2 -l tagset fst2
```

This program normalizes a text automaton `fst2`, according to the ELAG tag set `tagset`. It discards unknown codes and illegal entries. The text automaton is modified.

9.29 TextAutomaton2Mft

```
TextAutomaton2Mft text.fst2
```

This program takes a text automaton `text.fst2` as a parameter and constructs the equivalent in the `.mft` format of Intex. The produced file is called `text.mft` and is encoded in Unicode.

9.30 Tokenize

```
Tokenize text alphabet [-char_by_char]
```

This program cuts the text into word-like units. The `text` parameter represents the complete path of the text file, without omitting the `.snt` extension. The `alphabet` parameter represents the complete path of the alphabet definition file of the language of the text. The `-char_by_char` optional parameter indicates whether the program is applied character by character, with the exception of sentence separator `{S}` which is considered to be a single unit. Without this parameter, the program considers a unit to be either a sequence of letters (the letters are defined by file `alphabet`), or a character which is not a letter, or the sentence separator `{S}`, or a lexical label (`{aujourd'hui,.ADV}`).

The program codes each unit as a whole. The list of units is saved in a text file called `tokens.txt`. The sequence of codes representing the units now allows the coding of the text. This sequence is saved in a binary file named `text.cod`. The program also produces the following four files:

- `tok_by_freq.txt`: text file containing the units sorted by frequency;
- `tok_by_alph.txt`: text file containing the units sorted alphabetically;
- `stats.n`: text file containing information on the number of sentence separators, the number of units, the number of simple words and the number of numbers;

- `enter.pos`: binary file containing the list of newline positions in the text. The coded representation of the text does not contain newlines, but spaces. Since a newline counts as two characters and a space as a single one, it is necessary to know where newlines occur in the text when the positions of occurrences located by the `Locate` program are to be synchronized with the text file. File `enter.pos` is used for this by the `Concord` program. Thanks to this, when clicking on an occurrence in a concordance, it is correctly selected in the text. File `enter.pos` is a binary file containing the list of the positions of newlines in the text.

All produced files are saved in the directory of the text.

9.31 Txt2Fst2

```
Txt2Fst2 text alphabet [-clean] [norm]
```

This program constructs an automaton of a text. The `text` parameter represents the complete path of a text file without omitting the `.snt` extension. The `alphabet` parameter represents the complete path of the alphabet file of the language of the text. The `-clean` optional parameter indicates whether the rule of conservation of the best paths (see section 7.2.4) should be applied. If the `norm` parameter is specified, it is interpreted as the name of a normalization grammar that is to be applied to the text automaton.

If the text is separated into sentences, the program constructs an automaton for each sentence. If this is not the case, the program arbitrarily cuts the text into sequences of 2000 lexical units and produces an automaton for each of these sequences.

The result is a file called `text.fst2` which is saved in the directory of the text.

Chapter 10

File formats

This chapter presents the formats of files read or generated by Unitex. The formats of the DELAS and DELAF dictionaries have already been presented in sections 3.1.1 and 3.1.2.

NOTE: in this chapter the symbol ¶ represents the newline symbol. Unless otherwise indicated, all text files described in this chapter are encoded in Unicode Little-Endian.

10.1 Unicode Little-Endian encoding

All text files processed by Unitex have to be encoded in Unicode Little-Endian. This encoding allows the representation of 65536 characters by coding each of them in 2 bytes. In Little-Endian, the bytes are in lo-byte hi-byte order. If this order is reversed, we speak of Big-Endian. A text file encoded in Unicode Little-Endian starts with the special character with the hexadecimal value FEFF. The newline symbols have to be encoded by the two characters 000D and 000A.

Consider the following text:

Unitex¶
β-version¶

Here is its representation in Unicode Little-Endian:

header	U	n	i	t	e	x	¶	β
FFFE	5500	6E00	6900	7400	6500	7800	0D000A00	B203
-	v	e	r	s	i	o	n	¶
2D00	7600	6500	7200	7300	6900	6F00	6E00	0D000A00

Table 10.1: Hexadecimal representation of a Unicode text

The hi-bytes and lo-bytes have been reversed, which explains why the start character is encoded as FFFE in stead of FEFF, and 000D and OO0A are 0D00 and 0A00 respectively.

10.2 Alphabet files

There are two kinds of alphabet files: a file which defines the characters of a language, and a file that indicates the sorting preferences. The first is designed under the name *alphabet*, the second under the name *sorted alphabet*.

10.2.1 Alphabet

The alphabet file is a text file that describes all characters of a language, as well as the correspondances between capitalized and non-capitalized letters. This file is called `Alphabet.txt` and is found in the root of the directory of a language. Its presence is obligatory for Unitex to function.

Example: the English alphabet file has to be in the directory `.../English/`

Each line of the alphabet file must have one of the following three forms, followed by a newline symbol:

- #가힣 : a hash symbol followed by two characters X and Y which indicate that all characters between X and Y are letters. All these characters are considered to be in non-capitalized and capitalized form at the same time. This method is used to define the alphabets of Asian languages like Korean, Chinese or Japanese where there is no distinction between upper- and lower-case, and where the number of characters makes a complete enumeration tedious;
- Ëë : two characters X and Y indicate that X and Y are letters and that X is a capitalized equivalent of the non-capitalized Y form.
- 𠀀 : a unique character X defines X as a letter in capitalized and non-capitalized form. This form is used to define a single Asian character.

For certain languages like French, it is possible that a lower-case letter corresponds to multiple upper-case letters. For example, é, in practice, can have the upper-case form E or É. To express this, it suffices to use multiple lines. The reverse is equally true: a capitalized letter can correspond to multiple lower-case letters. Thus, E can be the capitalization of e, é, è, ê or ë. Here is an excerpt of the French alphabet file which defines different properties of letter e:

```
Ee¶
Eé¶
Éé¶
Eè¶
Èè¶
Eê¶
Êê¶
Eë¶
Ëë¶
```

10.2.2 Sorted alphabet

The sorted alphabet file defines the sorting priorities of the letters of a language. It is used by the `SortTxt` program. Each line of that file defines a group of letters. If a group of letters A is defined before a group of letters B, every letter of group A is inferior to every letter in group B.

The letters of a group are only distinguished if necessary. For example if the group of letters `eéèêë` has been defined, the word `ébahi` should be considered 'smaller' than `estuaire`, and also 'smaller' than `été`. Since the letters that follow `e` and `é` determine the order of the words, it is not necessary to compare letters `e` and `é` since they are of the same group. On the other hand, if the words `chantés` and `chantes` are to be sorted, `chantes` should be considered as 'smaller'. It is therefore necessary to compare the letters `e` and `é` to distinguish these words. Since the letter `e` appears first in the group `eéèêë`, it is considered to be 'smaller' than `chantés`. The word `chantes` should therefore be considered to be 'smaller' than the word `chantés`.

The sorted alphabet file allows the definition of equivalent characters. It is therefore possible to ignore the different accents as well as capitalization. For example, if the letters `b`, `c`, and `d` are to be ordered without considering capitalization and the cedilla, it is possible to write the following lines:

```
Bb¶
CcÇç¶
Dd¶
```

This file is optional. If no sorted alphabet file is specified, the `SortTxt` program sorts in the order of the Unicode encoding.

10.3 Graphs

This section presents the two graph formats: the graphic format `.grf` and the compiled format `.fst2`.

10.3.1 Format .grf

A `.grf` file is a text file that contains presentation information in addition to information representing the contents of the boxes and the transitions of the graph. A `.grf` file begins with the following lines:

```
#Unigraph¶
SIZE 1313 950¶
FONT Times New Roman:  12¶
OFONT Times New Roman:B 12¶
BCOLOR 16777215¶
FCOLOR 0¶
ACOLOR 12632256¶
```

```
SCOLOR 16711680¶
CCOLOR 255¶
DBOXES y¶
DFRAME y¶
DDATE y¶
DFILE y¶
DDIR y¶
DRIG n¶
DRST n¶
FITS 100¶
PORIENT L¶
#¶
```

The first line `#Unigraph` is a comment line. The following lines define the parameter values of the graph presentation:

- `SIZE x y` : defines the width `x` and the hight `y` of a graph in pixels;
- `FONT name:xyz` : defines the font used for displaying the contents of the boxes. `name` represents the name of the mode. `x` indicates if the text should be in bold face or not. If `x` is `B`, it indicates that it should be bold. For non-bold face, `x` should be a space. In the same way, `y` has value `I` if the text should be italic, a space if not. `z` represents the size of the text;
- `OFONT name:xyz` : defines the mode used for displaying transducer outputs. Parameters `name`, `x`, `y`, and `z` are defined in the same way as `FONT`;
- `BCOLOR x` : defines the background color of the graph. 'x' represents the color in RGB format;
- `FCOLOR x` : defines the foreground color of the graph. 'x' represents the color in RGB format;
- `ACOLOR x` : defines the color inside the boxes that correspond to the calls of subgraphs. `x` represents the color in RGB format;
- `SCOLOR x` : defines the color used for writing in comment boxes (boxes that are not linked up with any others). `x` represents the color in RGB format;
- `CCOLOR x` : defines the color used for designing selected boxes. `x` represents the color in RGB format;
- `DBOXES x` : this line is ignored by Unitex. It is conserved to ensure compatibility with Intex graphs;
- `DFRAME x` : there will be a frame around the graph if `x` is `y`, not if it is `n`;
- `DDATE x` : puts the date at the bottom of the graph if `x` is `y`, not if it is `n`;

- `DFILE x` : puts the name of the file at the bottom of the graph depending on whether `x` is `y` or `n`;
- `DDIR x` : prints the complete path of the graph wether `x` is `y` or `n`. This option has no effect if the `DFILE` option is set to `n`;
- `DRIG x` : displays the graph from right to left or left to right depending on whether `x` is `y` or `n`;
- `DRST x` : this line is ignored by Unitex. It isconserved to ensure compatibility with Intex graphs;
- `FITS x` : this line is ignored by Unitex. It isconserved to ensure compatibility with Intex graphs;
- `PORIENT x` : this line is ignored by Unitex. It isconserved to ensure compatibility with Intex graphs;
- `#` : this line is ignored by Unitex. It serves to indicate the end of the header information.

The lines after the header give the contents and the position of the boxes in the graph. The following example corresponds to a graph recognizing a number:

```
3¶
"<E>" 84 248 1 2 ¶
"" 272 248 0 ¶
s"1+2+3+4+5+6+7+8+9+0" 172 248 1 1 ¶
```

The first line after the header indicates the number of boxes in the graph, immediately followed by a newline. This number can not be lower than 2, since a graph always has an initial and a final state.

The following lines define the boxes of the graph. The boxes are numbered starting at 0. By convention, state 0 is the initial state and state 1 is the final state. The contents of the final state is always empty.

Each box in the graph is defined by a line that has the following format:

contents X Y N transitions ¶

contents is a sequence of characters enclosed in quotation marks that represents the contents of the box. This sequence can sometimes be preceded by an `s` if the graph is imported from Intex; this character is then ignored by Unitex. The contents of the sequence is the text that has been entered in the editing line of the graph editor. Table 10.2 shows the encoding of two special sequences that are not encoded in the same way as they are entered into the `.grf` files:

Sequence in the graph editor	Sequence in the `.grf` file
"	\"
\"	\\\"

Table 10.2: Encoding of special sequences

NOTE: The characters between < and > or between { and } are not interpreted. Thus the + character in sequence "`le <A+Conc>`" is not interpreted as a line separator, since the pattern `<A+Conc>` is interpreted with priority.

X and *Y* represent the coordinates of the box in pixels. Figure 10.1 shows how these coordinates are interpreted by Unitex.

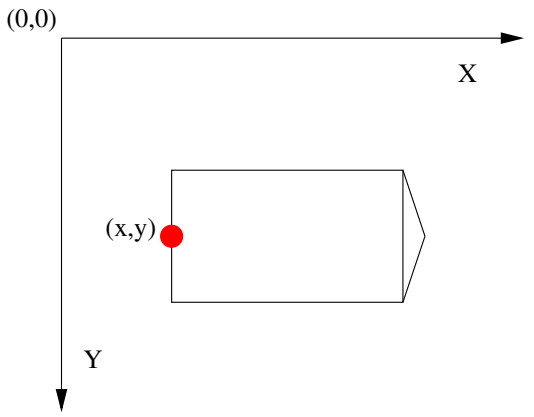


Figure 10.1: Interpretation of the coordinates of boxes

N represents the number of outgoing transitions of the box. This number is always 0 for the final state.

The transitions are defined by the number of their target box.

Every line of the box definition ends with a newline.

10.3.2 Format .fst2

An `.fst2` file is a text file that describes a set of graphs. Here is an example of an `.fst2` file:

```
0000000002¶
-1 NP¶
: 1 1 ¶
```

```
: 2 2 -2 2 ¶
: 3 3 ¶
t ¶
f ¶
-2 Adj¶
: 6 1 5 1 4 1 ¶
t ¶
f ¶
%<E>¶
%the/DET¶
%<A>/ADJ¶
%<N>¶
%nice¶
@pretty¶
%small¶
f¶
```

The first line represents the number of graphs that are encoded in the file. The beginning of each graph is identified by a line that indicates the number and the name of the graph (`-1 NP` and `-2 Adj` in the file above).

The following lines describe the states of the graph. If the state is final, the line starts with the `t` character and with the `:` character if not. For each state, the list of transitions is a possibly empty sequence of pairs of integers:

- the first integer indicates the number of the label or sub-graph that corresponds to the transition. Labels are numbered starting at 0. Sub-graphs are represented by negative integers, which explains why the numbers preceding the names of the graphs are negative;
- the second integer represents the number of the result state after the transition. In each graph, the states are numbered starting at 0. By convention state 0 is the initial state.

Each state definition line terminates with a space. The end of each graph is marked by a line containing an `f` followed by a space and a newline.

Labels are defined after the last graph. If the line begins with the `@` character, the contents of the label is to be searched without allowing case variations. This information is not used if the label is not a word. If the line starts with a `%`, capitalization variants are authorized. If a label carries a transducer output sequence, the input and output sequences are separated by the `/` character (example: `the/DET`). By convention, the first label is always the empty word (`<E>`), even if that label is never used for any transition.

The end of the file is indicated by a line containing the `f` character followed by a newline.

10.4 Texts

This section presents the different files used to represent texts.

10.4.1 .txt files

`.txt` files are text files encoded in Unicode Little-Endian. These files should not contain any opening or closing braces, except for those used to mark a sentence separator (`{S}`) or a valid lexical tag (`{aujourd'hui,.ADV}`). The newline needs to be encoded with the two special characters with hexadecimal values `000D` and `000A`.

10.4.2 .snt Files

`.snt` files are `.txt` files that have been processed by Unitex. These files should not contain any tabs. They should also not contain multiple consecutive spaces or newlines. The only allowed braces in `.snt` files are those of the sentence delimiter `{S}` and those of lexical labels (`{aujourd'hui,.ADV}`).

10.4.3 File text.cod

The `text.cod` file is a binary file containing a sequence of integers that represent the text. Each integer `i` reflects the token with index `i` in the `tokens.txt` file. These integers are encoded in four bytes.

NOTE: The tokens are numbered starting at 0.

10.4.4 The tokens.txt file

The `tokens.txt` file is a text file that contains the list of all lexical units of the text. The first line of this file indicates the number of units found in the file. Units are separated by a newline. Whenever a sequence is found in the text with capitalization variants, each variant is encoded as a distinct unit.

NOTE: newlines that might be in the `.snt` file are encoded like spaces. Therefore there is never a unit encoding the newline.

10.4.5 The tok_by_alph.txt and tok_by_freq.txt files

These two files are text files that contain the list of lexical units sorted alphabetically or by frequence.

In the `tok_by_alph.txt` file, each line is composed by a unit, followed by a tab and the number of occurrences of the unit within the text.

The lines of the `tok_by_freq.txt` file are formed after the same principle, but the number of occurrences is placed after the tab and the unit.

10.4.6 The enter.pos file

This file is a binary file containing the list of positions of the newline symbol in the `.snt` file. Each position is the index in the `text.cod` file where a newline has been replaced by a space. These positions are integers that are encoded in 4 bytes.

10.5 Text Automaton

10.5.1 The text.fst2 file

The `text.fst2` file is a special `.fst2` file that represents the text automaton. In that file, each sub-graph represents a sentence automaton. The areas reserved for the names of the sub-graphs are used to store the sentences from which the sentence automata have been constructed.

With the exception of the first label which is always the empty word (`<E>`), the labels have to be either lexical units or entries in the DELAF format in braces.

Example: Here is the file that corresponds to the text *He is drinking orange juice.*

```
0000000001¶
-1 He is drinking orange juice. ¶
: 1 1 2 1 ¶
: 3 2 4 2 ¶
: 5 3 6 3 7 3 ¶
: 8 4 9 4 10 4 11 5 ¶
: 12 5 13 5 ¶
: 14 6 ¶
t ¶
f ¶
%<E>¶
%{He,he.N:s:p}¶
%{He,he.PRO+Nomin:3ms}¶
%{is,be.V:P3s}¶
%{is,i.N:p}¶
%{drinking,drinking.A}¶
%{drinking,drinking.N:s}¶
%{drinking,drink.V:G}¶
%{orange,orange.A}¶
%{orange,orange.N+Conc:s}¶
%{orange,orange.N:s}¶
%{orange juice,orange juice.N+XN+z1:s}¶
%{juice,juice.N+Conc:s}¶
%{juice,juice.V:W:P1s:P2s:P1p:P2p:P3p}¶
%.¶
f¶
```

10.5.2 The cursentence.grf file

The `cursentence.grf` file is generated by Unitex during the display of a sentence automaton. The `Fst2Grf` program constructs a `.grf` file from the `text.fst2` file that represents a sentence automaton.

10.5.3 The sentenceN.grf file

Whenever the user modifies a sentence automaton, that automaton is saved under the name `sentenceN.grf`, where `N` represents the number of the sentence.

10.5.4 The cursentence.txt file

During the extraction of the sentence automaton, the text of the sentence is saved in the file called `cursentence.txt`. That file is used by Unitex to display the text of the sentence under the automaton. That file contains the text of the sentence, followed by a newline.

10.6 Concordances

10.6.1 The concord.ind file

The `concord.ind` file is the index of the occurrences found by the program `Locate` during the application of a grammar. It is a text file that contains the starting and ending position of each occurrence, possibly accompanied by a sequence of letters if the construction of the concordance took into account the possible transducer outputs of the grammar. Here is an example of such a file:

```
#M¶
59 63 the[ADJ= greater] part¶
67 71 the beautiful hills¶
87 91 the pleasant town¶
123 127 the noble seats¶
157 161 the fabulous Dragon¶
189 193 the Civil Wars¶
455 459 the feeble interference¶
463 467 the English Council¶
568 572 the national convulsions¶
592 596 the inferior gentry¶
628 632 the English constitution¶
698 702 the petty kings¶
815 819 the certain hazard¶
898 902 the great Barons¶
940 944 the very edge¶
```

The first line indicates in which transduction mode the concordance has been constructed. The three possible values are:

- `#I` : transducer outputs have been ignored;
- `#M` : transducer outputs have been inserted before the corresponding inputs (MERGE mode);
- `#R` : transducer outputs have replaced the recognized sequences (REPLACE mode)).

Each occurrence is described in one line. The lines start with the start and end position of the occurrence. These positions are given in lexical units.

If the file has the heading line `#I`, the end position of each occurrence is immediately followed by a newline. Otherwise, it is followed by a space and a sequence of characters. In REPLACE mode, that sequence corresponds to the output produced for the recognized sequence. In MERGE mode, it represents the recognized sequences into which the outputs have been inserted. In MERGE or REPLACE mode, this sequence is displayed in the concordance. If the outputs have been ignored, the contents of the occurrence is extracted from the text file.

10.6.2 The concord.txt file

The `concord.txt` file is a text file that represents a concordance. Each occurrence is encoded in a line that is composed of three character sequences separated by a tab, representing the left context, the occurrence (possibly modified by transducer outputs) and the right context.

10.6.3 The concord.html file

The `concord.html` file is an `HTML` file that represents a concordance. This file is encoded in UTF-8.

The title of the page is the number of occurrences it describes. The lines of the concordance are encoded as lines where the occurrences are considered to be hypertext lines. The reference associated to each of these lines has the following form: `<a href="X Y Z">`. `X` and `Y` represent the start and end position of the occurrence in characters in the file `name_of_text.snt`. `Z` represents the number of the phrase in which this occurrence appears.

All spaces that are at the left and right edges of lines are encoded by a non breaking space (` ` in HTML), which allows the preservation of the alignment of the utterances even if one of them has a left context with spaces.

NOTE: if the concordance has been constructed with the `glossanet` parameter, the HTML file obtains the same structure, except for the links. In these concordances, the occurrences are real links pointing at the web server of the GlossaNet application. For more information on GlossaNet, consult the link on the Unitex web site.

Here is an example of a file:

```
<html lang=en>¶
<head>¶
¶
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">¶
   <title>6 matches</title>¶
</head>¶
<body>¶
<table border="0" width="100%"><td nowrap>¶
<font face="Courier new" size=3>¶
on, there <a href="116 124 2">extended</a> i <br>¶
 extended <a href="125 127 2">in</a> ancient <br>¶
 Scott {S}<a href="32 34 2">IN</a> THAT PL <br>¶
STRICT of <a href="61 66 2">merry</a> Engl <br>¶
S}IN THAT <a href="40 48 2">PLEASANT</a> D <br>¶
 which is <a href="84 91 2">watered</a> by <br>¶
</font>¶
</td></table></body>¶
</html>¶
```

Figure 10.2 shows the page that corresponds to the file below.

Figure 10.2: Example of a concordance

10.6.4 The diff.html file

The `diff.html` file is an `HTML` file that presents the differences between two concordances. This file is encoded in UTF-8. Here is an example of file (new lines have been introduced for presentation convenience):

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html;
         charset=UTF-8">
   <style type="text/css">
```

```
   a.blue {color:blue; text-decoration:underline;}
   a.red {color:red; text-decoration:underline;}
   a.green {color:green; text-decoration:underline;}
   </style>
</head>
<body>
<h4>
<font color="blue">Blue:</font> identical sequences<br>
<font color="red">Red:</font> similar but different sequences<br>
<font color="green">Green:</font> sequences that occur in only
one of the two concordances<br>
<table border="1" cellpadding="0" style="font-family: Courier new;
  font-size: 12">
<tr><td width="450"><font color="blue">ed in ancient times
      <u>a large forest</u>, covering the greater par</font></td>
    <td width="450"><font color="blue">ed in ancient times
      <u>a largeforest</u>, covering the greater par</font></td>
</tr>
<tr><td width="450"><font color="green">ge forest, covering
      <u>the greater part</u> of the beautiful hills </font>
      </td>
      <td width="450"><font color="green"></font></td>
</tr>
</table>
</body>
</html>
```

10.7 Dictionaries

The compression of the DELAF dictionaries by the `Compress` program produces two files: a `.bin` file that represents the minimal automaton of the inflected forms of the dictionaries, and a `.inf` file that contains the compressed forms required for the construction of the dictionaries from the inflected forms. This section describes the format of these two file types, as well as the format of the `CHECK_DIC.TXT` file, which contains the result of the verification of a dictionary.

10.7.1 The .bin files

A `.bin` file is a binary file that represents an automaton. The first 4 bytes of the file represent an integer that indicates the size of the file in bytes. The states of the automaton are encoded in the following way:

- the first two bytes indicate if the state is final as well as the number of its outgoing transitions. The highest bit is 0 if the state is final, 1 if not. The other 15 bits encode the number of transitions. Example: a non-final state with 17 transitions is encoded by the

hexadecimal sequence 8011

- if the state is final, the three following bytes encode the index in the `.inf` file of the compressed form to be used to reconstruct the dictionary lines for this inflected form.

  Example: if the state refers to the compressed form with index 25133, the corresponding hexadecimal sequence is `00622D`

- each leaving transition is then encoded in 5 bytes. The first 2 bytes encode the character that labels the transition, and the three following encode the byte position of the result state in the `.bin` file. The transitions of a state are encoded next to each other.

  Example: a transition that is labeled with the `A` letter and goes to the state of which the description starts at byte 50106, is represented by the hexadecimal sequence `004100C3BA`.

By convention, the first state of the automaton is the initial state.

10.7.2 The .inf files

A `.inf` file is a text file that describes the compressed files that are associated to a `.bin` file. Here an example of a `.inf` file:

```
0000000006¶
_10\0\0\7.N¶
.PREP¶
_3.PREP¶
.PREP,_3.PREP¶
1-1.N+Hum:mp¶
3er 1.N+AN+Hum:fs¶
```

The first line of the file indicates the number of compressed forms that it contains. Each line can contain one or more compressed forms. If there are multiple forms, they are separated by commas. Each compressed form is made up of a sequence required to reconstruct a canonical knowing an inflected form, followed by a sequence of grammatical, semantic and inflection codes that are associated to the entry.

The mode of compression of the canonical form varies in function of the inflected form. If the two forms are identical, the compressed form contains only the grammatical, semantic and inflectional information as in:

```
.N+Hum:ms
```

If the forms are different, the compression program cuts up the two forms in units. These units can be a space, a hyphen, or a sequence of characters that contains neither a space nor

a hyphen. This way of cutting up units allows the program to efficiently take into account the inflected forms of the compound words.

If the inflected and the canonical form do not have the same number of units, the program encodes the canonical form by the number of characters to be removed from the inflected form followed by the characters to append. For instance, the line below is a line in the initial dictionary:

```
James Bond,007.N
```

Since the sequence `James Bond` contains three units and `007` only one, the canonical form is encoded with `_10\0\0\7`. The `_` character indicates that the two forms do not have the same number of units. The following number (here 10) indicates the number of characters to be removed. The sequence `\0\0\7` indicates that the sequence `007` should be appended. The digits are preceeded by the `\` character so they will not be confused with the number of characters to be removed.

Whenever the two forms have the same number of units, the units are compressed two by two. Each pair consists of a unit the inflected form and the corresponding unit in the canonical form. If each of the two units is a space or a hyphen, the compressed form of the unit is the unit itself, as in the following line:

```
0-1.N:p
```

which is the output for `battle-axes,battle-axe.N:p`

This maintains a certain readability of the `.inf` file when the dictionary contains compound words.

Whenever one or both of the units in a pair is neither a space nor a hyphen, the compressed form is composed of the number of characters to be removed followed by the sequence of characters to be appended. Thus, the dictionary line:

```
première partie,premier parti.N+AN+Hum:fs
```

is encoded by the line:

```
3er 1.N+AN+Hum:fs
```

The `3er` code indicates that 3 characters are to be removed from the sequence `première` and the characters `er` are to be appended to obtain `premier`. The `1` indicates that only one character needs to be removed from `partie` to obtain `parti`. The number `0` is used whenever it needs to be indicated that no letter should be removed.

10.7.3 The CHECK_DIC.TXT file

This file is produced by the dictionary verification program `CheckDic`. It is a text file that contains information about the analysed dictionary and has four parts.

The first part is the possibly empty list of all syntax errors found in the dictionary: absence of the inflected or the canonical form, the grammatical code, empty lines, etc. Each error is described by the number of the line, a message describing the error, and the contents of the line. Here is an example of a message:

```
Line 12451: no point found
garden,N:s
```

The second and third parts display the list of grammatical codes and/or semantic and inflectional codes respectively. In order to prevent coding errors, the program reports encodings that contain spaces, tabs, or non-ASCII characters. For instance, if a Greek dictionary contains the `ADV` code where the Greek `A` character is used instead of the Latin `A` character, the program reports the following warning:

```
ADV warning: 1 suspect char (1 non ASCII char): (0391 D V)
```

Non-ASCII characters are indicated by their hexadecimal character number. In the example below, the code `0391` represents Greek `A`. Spaces are indicated by the `SPACE` sequence:

```
Km s warning: 1 suspect char (1 space): (K m SPACE s)
```

When the following dictionary is verified:

```
1,2 et 3!,.INTJ
abracadabra,INTJ
supercalifragilisticexpialidocious,.INTJ
damned,. INTJ
```

the following `CHECK_DIC.TXT` file is obtained:

```
Line 1: unprotected comma in lemma¶
1,2 et 3!,.INTJ¶
Line 2: no point found ¶
abracadabra,INTJ ¶
---------------------------------- ¶
----  All chars used in forms  ---- ¶
---------------------------------- ¶
  (0020) ¶
! (0021) ¶
, (002C) ¶
1 (0031) ¶
```

```
2 (0032) ¶
3 (0033) ¶
I (0049) ¶
J (004A) ¶
N (004E) ¶
T (0054) ¶
a (0061) ¶
b (0062) ¶
c (0063) ¶
d (0064) ¶
e (0065) ¶
f (0066) ¶
g (0067) ¶
i (0069) ¶
l (006C) ¶
m (006D) ¶
n (006E) ¶
o (006F) ¶
p (0070) ¶
r (0072) ¶
s (0073) ¶
t (0074) ¶
u (0075) ¶
x (0078) ¶
------------------------------------------------------------ ¶
----    2 grammatical/semantic codes used in dictionary  ---- ¶
------------------------------------------------------------ ¶
INTJ ¶
 INTJ warning: 1 suspect char (1 space): (SPACE I N T J) ¶
------------------------------------------------------ ¶
----    0 inflectional code used in dictionary  ----- ¶
------------------------------------------------------ ¶
```

10.8 ELAG files

10.8.1 tagset.def file

See section 7.3.6, page 125.

10.8.2 .lst files

.LST FILES ARE NOT UNICODE FILES.

A `.lst` file contains a list of `.grf` file names. These files are supposed to be located in the ELAG directory corresponding to the current working language. Here is the `elag.lst` file used for French:

```
PPVs/PpvIL.grf¶
PPVs/PpvLE.grf¶
PPVs/PpvLUI.grf¶
PPVs/PpvPR.grf¶
PPVs/PpvSeq.grf¶
PPVs/SE.grf¶
PPVs/postpos.grf¶
```

10.8.3 .elg files

`.elg` files contain compiled ELAG rules. These files are in the `.fst2` format.

10.8.4 .rul files

.RUL FILES ARE NOT UNICODE FILES.

A `.rul` file contains the different `.elg` files that compose an ELAG rule set. It contains one part per `.elg` file. Each part lists the ELAG grammars that correspond to a given `.elg` file. `.elg` file names are surrounded with angles brackets. The lines that start with a tabulation are considered as comments by the `Elag` program. Here is the `elag.rul` file used for French:

```
    PPVs/PpvIL.elg¶
    PPVs/PpvLE.elg¶
    PPVs/PpvLUI.elg¶
<elag.rul-0.elg>¶
    PPVs/PpvPR.elg¶
    PPVs/PpvSeq.elg¶
    PPVs/SE.elg¶
    PPVs/postpos.elg¶
<elag.rul-1.elg>¶
```

10.9 Configuration files

10.9.1 The Config file

Whenever the user modifies his preferences for a given languages, these modifications are saved in a text file named 'Config' which can be found in the directory of the current language. The file has the following syntax (the order of lines can vary):

```
#Unitex configuration file of 'paumier' for 'English'¶
#Tue Jan 31 11:21:32 CET 2006¶
```

```
TEXT\ FONT\ NAME=Courier New¶
TEXT\ FONT\ STYLE=0¶
TEXT\ FONT\ SIZE=10¶
CONCORDANCE\ FONT\ NAME=Courier new¶
CONCORDANCE\ FONT\ HTML\ SIZE=12¶
INPUT\ FONT\ NAME=Times New Roman¶
INPUT\ FONT\ STYLE=0¶
INPUT\ FONT\ SIZE=10¶
OUTPUT\ FONT\ NAME=Arial Unicode MS¶
OUTPUT\ FONT\ STYLE=1¶
OUTPUT\ FONT\ SIZE=12¶
DATE=true¶
FILE\ NAME=true¶
PATH\ NAME=false¶
FRAME=true¶
RIGHT\ TO\ LEFT=false¶
BACKGROUND\ COLOR=-1¶
FOREGROUND\ COLOR=-16777216¶
AUXILIARY\ NODES\ COLOR=-3289651¶
COMMENT\ NODES\ COLOR=-65536¶
SELECTED\ NODES\ COLOR=-16776961¶
PACKAGE\ NODES\ COLOR=-2302976¶
CONTEXT\ NODES\ COLOR=-16711936¶
CHAR\ BY\ CHAR=false¶
ANTIALIASING=false¶
HTML\ VIEWER=¶
MAX\ TEXT\ FILE\ SIZE=2097152¶
ICON\ BAR\ POSITION=West¶
PACKAGE\ PATH=D\:\\repository¶
```

The first two lines are comment lines. The following three lines indicate the name, the style and the size of the font used to display texts, dictionaries, lexical units, sentences in text automata, etc.

The `CONCORDANCE FONT NAME` and `CONCORDANCE FONT HTML SIZE` parameters define the name, the size and the font to be used when displaying concordances in HTML. The size of the font has a value between 1 and 7.

The `INPUT FONT ...` and `OUTPUT FONT ...` parameters define the name, the style and the size of the fonts used for displaying the paths and the transducer outputs of the graphs.

The following 10 parameters correspond to the parameters given in the headings of the graphs. Table 10.3 describes the correspondances.

Parameters in the Config file	Parameters in the .grf file
DATE	DDATE
FILE NAME	DFILE
PATH NAME	DDIR
FRAME	DFRAME
RIGHT TO LEFT	DRIG
BACKGROUND COLOR	BCOLOR
FOREGROUND COLOR	FCOLOR
AUXILIARY NODES COLOR	ACOLOR
COMMENT NODES COLOR	SCOLOR
SELECTED NODES COLOR	CCOLOR

Table 10.3: Meaning of the parameters

The PACKAGE NODES parameter defines the color to be used for displaying calls to subgraphs located in the repository.

The CONTEXT NODES parameter defines the color to be used for displaying boxes that correspond to context bounds.

The CONTEXT NODES indicates if the current language must be tokenized character by character or not.

The ANTIALIASING parameter indicates whether graphs as well as sentence automata are displayed by default with the antialiasing effect.

The HTML VIEWER parameter indicates the name of the navigator to be used for displaying concordances. If no navigator name is defined, concordances are displayed in a Unitex window.

The MAX TEXT FILE SIZE parameter indicates the maximum size in bytes of text files that can be viewed in Unitex. If a file size is upper than this value, the message " This file is too large to be displayed. Use a wordprocessor to view it." will be shown. The default value is 1024000 (1000 Kbytes).

The ICON BAR POSITION parameter indicates the default position of icon bars in graph frames.

The PACKAGE PATH parameter specifies the location of the repository.

10.9.2 The system_dic.def file

The system_dic.def file is a text file that describes the list of system dictionaries that are applied by default. This file can be found in the directory of the current language. Each line corresponds to a name of a .bin file. The system dictionaries are in the system directory, and

in that directory in the `(current language)/Dela` sub-directory. Here is an example of this file:

```
delacf.bin¶
delaf.bin¶
```

10.9.3 The user_dic.def file

The `user_dic.def` file is a text file that describes the list of dictionaries the user has defined to be applied by default. This file is in the directory of the current language and has the same format as the `system_dic.def` file. The dictionaries need to be in the `(current language)/Dela` sub-directory of the personal directory of the user.

10.9.4 The user.cfg file

Under Linux, Unitex expects the personal directory of the user to be called `unitex` and expects it to be in his root directory (`$HOME`). Under Windows, it is not always possible to associate a directory to a user per default. To compensate for that, Unitex creates a `.cfg` file for each user that contains the path to his personal directory. This file is saved under the name `(user login).cfg` in the `Unitex/Users` system sub-directory.

WARNING: THIS FILE IS NOT IN UNICODE AND THE PATH OF THE PERSONAL DIRECTORY IS NOT FOLLOWED BY A NEWLINE.

10.10 Various other files

For each text, Unitex creates multiple files that contain information that are designed to be displayed in the graphical interface. This section describes these files.

10.10.1 The dlf.n, dlc.n et err.n files

These three files are text files that are stored in the text directory. They contain the number of lines of the `dlf`, `dlc` and `err` files respectively. These numbers are followed by a newline.

10.10.2 The stat_dic.n file

This file is a text file in the directory of the text. It has three lines that contain the number of lines of the `dlf`, `dlc` and `err` files.

10.10.3 The stats.n file

This file is in the text directory and contains a line with the following form:

```
3949 sentence delimiters, 169394 (9428 diff) tokens, 73788 (9399) simple
 forms, 438 (10) digits¶
```

The numbers indicated are interpreted in the following way:

- `sentence delimiters`: number of sentence separators (`{S}`);
- `tokens`: total number of lexical units in the text. The number preceeding `diff` indicates the number of different units;
- `simple forms`: the total number of lexical units in the text that are composed of letters. The number in parentheses represents the number of different lexical units that are composed of letters;
- `digits`: the total number of digits used in the text. The number in parentheses indicates the number of different digits used (10 at most).

10.10.4 The concord.n file

The `concord.n` file is a text file in the directory of the text. It contains information on the latest search of the text and looks like the following:

```
6 matches¶
6 recognized units¶
(0.004% of the text is covered)¶
```

The first line gives the number of found occurrences, and the second the name of units covered by these occurrences. The third line indicates the ratio between the covered units and the total number of units in the text.

Appendix A - GNU General Public License

This license can also be found in [23].

Version 2, June 1991

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B - GNU Lesser General Public License

This license can also be found in [24].

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive

or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses

the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore

falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do

not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice
That's all there is to it!

Appendix C - Lesser General Public License For Linguistic Resources

This license was designed by the University of Marne-la-Vallée, and it has received the approval of the Free Software Foundation ([1]).

Preamble

The licenses for most data are designed to take away your freedom to share and change it. By contrast, this License is intended to guarantee your freedom to share and change free data–to make sure the data are free for all their users.

This license, the Lesser General Public License for Linguistic Resources, applies to some specially designated linguistic resources – typically lexicons, grammars, thesauri and textual corpora.

Terms and Conditions For Copying, Distribution and Modification

0. This License Agreement applies to any Linguistic Resource which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License for Linguistic Resources (also called "this License"). Each licensee is addressed as "you".

   A "linguistic resource" means a collection of data about language prepared so as to be used with application programs.

   The "Linguistic Resource", below, refers to any such work which has been distributed under these terms. A "work based on the Linguistic Resource" means either the Linguistic Resource or any derivative work under copyright law: that is to say, a work containing the Linguistic Resource or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Legible form" for a linguistic resource means the preferred form of the resource for making modifications to it.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Linguistic

Resource is not restricted, and output from such a program is covered only if its contents constitute a work based on the Linguistic Resource (independent of the use of the Linguistic Resource in a tool for writing it). Whether that is true depends on what the program that uses the Linguistic Resource does.

1. You may copy and distribute verbatim copies of the Linguistic Resource as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Linguistic Resource.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Linguistic Resource or any portion of it, thus forming a work based on the Linguistic Resource, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) The modified work must itself be a linguistic resource.

   (b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   (c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Linguistic Resource, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Linguistic Resource, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Linguistic Resource.

   In addition, mere aggregation of another work not based on the Linguistic Resource with the Linguistic Resource (or with a work based on the Linguistic Resource) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. A program that contains no derivative of any portion of the Linguistic Resource, but is designed to work with the Linguistic Resource (or an encrypted form of the Linguistic Resource) by reading it or being compiled or linked with it, is called a "work that uses

the Linguistic Resource". Such a work, in isolation, is not a derivative work of the Linguistic Resource, and therefore falls outside the scope of this License.

However, combining a "work that uses the Linguistic Resource" with the Linguistic Resource (or an encrypted form of the Linguistic Resource) creates a package that is a derivative of the Linguistic Resource (because it contains portions of the Linguistic Resource), rather than a "work that uses the Linguistic Resource". If the package is a derivative of the Linguistic Resource, you may distribute the package under the terms of Section 4. Any works containing that package also fall under Section 4.

4. As an exception to the Sections above, you may also combine a "work that uses the Linguistic Resource" with the Linguistic Resource (or an encrypted form of the Linguistic Resource) to produce a package containing portions of the Linguistic Resource, and distribute that package under terms of your choice, provided that the terms permit modification of the package for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the package that the Linguistic Resource is used in it and that the Linguistic Resource and its use are covered by this License. You must supply a copy of this License. If the package during execution displays copyright notices, you must include the copyright notice for the Linguistic Resource among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   (a) Accompany the package with the complete corresponding machine-readable legible form of the Linguistic Resource including whatever changes were used in the package (which must be distributed under Sections 1 and 2 above); and, if the package contains an encrypted form of the Linguistic Resource, with the complete machine-readable "work that uses the Linguistic Resource", as object code and/or source code, so that the user can modify the Linguistic Resource and then encrypt it to produce a modified package containing the modified Linguistic Resource.

   (b) Use a suitable mechanism for combining with the Linguistic Resource. A suitable mechanism is one that will operate properly with a modified version of the Linguistic Resource, if the user installs one, as long as the modified version is interface-compatible with the version that the package was made with.

   (c) Accompany the package with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 4a, above, for a charge no more than the cost of performing this distribution.

   (d) If distribution of the package is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   (e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

If the package includes an encrypted form of the Linguistic Resource, the required form of the "work that uses the Linguistic Resource" must include any data and utility programs needed for reproducing the package from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Linguistic Resource together in a package that you distribute.

5. You may not copy, modify, sublicense, link with, or distribute the Linguistic Resource except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Linguistic Resource is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Linguistic Resource or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Linguistic Resource (or any work based on the Linguistic Resource), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Linguistic Resource or works based on it.

7. Each time you redistribute the Linguistic Resource (or any work based on the Linguistic Resource), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Linguistic Resource subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Linguistic Resource at all. For example, if a patent license would not permit royalty-free redistribution of the Linguistic Resource by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Linguistic Resource.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole

is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free resource distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of data distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute resources through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Linguistic Resource is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Linguistic Resource under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License for Linguistic Resources from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Linguistic Resource specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Linguistic Resource does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Linguistic Resource into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission.

NO WARRANTY

12. BECAUSE THE LINGUISTIC RESOURCE IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LINGUISTIC RESOURCE, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LINGUISTIC RESOURCE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LINGUISTIC RESOURCE IS WITH YOU. SHOULD

THE LINGUISTIC RESOURCE PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LINGUISTIC RESOURCE AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LINGUISTIC RESOURCE (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LINGUISTIC RESOURCE TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Bibliography

[1] Free Software Foundation. http://www.fsf.org. 10.10.4

[2] Anna ANASTASSIADIS-SYMEONIDIS, Tita KYRIACOPOULOU, Elsa SKLAVOUNOU, Iasson THILIKOS, and Rania VOSKAKI. A system for analysing texts in modern greek: representing and solving ambiguities. In *Proceedings of COMLEX 2000, Workshop on Computational Lexicography and Multimedia Dictionaries*. Patras, 2000. 3.7

[3] Olivier BLANC and Anne DISTER. Automates lexicaux avec structure de traits. 2004. Actes RECITAL 2004. 7.3

[4] Xavier BLANCO. Noms composés et traduction français-espagnol. *Lingvisticæ Investigationes*, 21(1), 1997. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[5] Xavier BLANCO. Les dictionnaires électroniques de l'espagnol (DELASs et DELACs). *Lingvisticæ Investigationes*, 23(2), 2000. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[6] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. La structure des phrases simples en français : classes de constructions transitives. Technical report, LADL, Paris, 1976. 8.1

[7] Jean-Paul BOONS, Alain GUILLET, and Christian LECLÈRE. *La structure des phrases simples en français : constructions intransitives*. Droz, Genève, 1976. 8.1

[8] Firefox. Web browser. http://www.mozilla.com/firefox/. 4.8.2

[9] Netscape. Web browser. http://www.netscape.com. 4.8.2

[10] Folker CAROLI. Les verbes transitifs à complément de lieu en allemand. *Lingvisticæ Investigationes*, 8(2):225–267, 1984. Amsterdam-Philadelphia : John Benjamins Publishing Company. 8.1

[11] A. CHROBOT, B. COURTOIS, M. HAMMANI-MC CARTHY, M. GROSS, and K. ZELLAGUI. Dictionnaire electronique DELAC anglais : noms composés. Technical Report 59, LADL, Université Paris 7, 1999. 3.7

[12] Unicode Consortium. http://www.unicode.org. 2.2

[13] Matthieu CONSTANT and Anastasia YANNACOPOULOU. Le dictionnaire électronique du grec moderne: Conception et développement d'outils pour son enrichissement et sa validation. In *Studies in Greek Linguistics, Proceedings of the 23rd annual meeting of the Department of Linguistics*. Faculty of Philosophy, Aristotle University of Thessaloniki, 2002. 3.7

[14] Blandine COURTOIS. Formes ambiguës de la langue française. *Lingvisticæ Investigationes*, 20(1):167–202, 1996. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[15] Blandine Courtois and Max Silberztein, editors. *Les dictionnaires électroniques du français*. Larousse, Langue française, vol. 87, 1990. 3.7

[16] Anne DISTER, Nathalie FRIBURGER, and Denis MAUREL. Améliorer le découpage en phrases sous INTEX. In Anne Dister, editor, *Revue Informatique et Statistique dans les Sciences Humaines*, volume Actes des 3èmes Journées INTEX, pages 181–199, 2000. 2.5.2

[17] Anibale ELIA. *Le verbe italien. Les complétives dans les phrases à un complément*. Schena/Nizet, Fasano/Paris, 1984. 8.1

[18] Anibale ELIA. *Lessico-grammatica dei verbi italiani a completiva. Tavole e indice generale*. Liguori, Napoli, 1984. 8.1

[19] Anibale ELIA and Simoneta VIETRI. Electronic dictionaries and linguistic analysis of italian large corpora. In *Actes des 5es Journées internationales d'Analyse statistique des Données Textuelles*. Ecole Polytechnique fédérale de Lausanne, 2000. 3.7

[20] Anibale ELIA and Simoneta VIETRI. L'analisi automatica dei testi e i dizionari elettronici. In E. Burattini and R. Cordeschi, editors, *Manuale di Intelligenza Artificiale per le Scienze Umane*. Roma:Carocci, 2002. 3.7

[21] Jacqueline GIRY-SCHNEIDER. *Les nominalisations en français. L'opérateur faire dans le lexique*. Droz, Genève-Paris, 1978. 8.1

[22] Jacqueline GIRY-SCHNEIDER. *Les prédicats nominaux en français. Les phrases simples à verbe support*. Droz, Genève-Paris, 1987. 8.1

[23] GNU. General Public License. http://www.gnu.org/licenses/gpl.html. 1.1, 10.10.4

[24] GNU. Lesser General Public License. http://www.gnu.org/licenses/lgpl.html. 1.1, 10.10.4

[25] Gaston GROSS. *Les expressions figées en français*. Ophrys, Paris, 1996. 3.7

[26] Maurice GROSS. *Méthodes en syntaxe*. Hermann, Paris, 1975. 8.1

[27] Maurice GROSS. *Grammaire transformationnelle du français. 3 - Syntaxe de l'adverbe*. ASSTRIL, Paris, 1986. 3.7, 8.1

[28] Alain GUILLET and Christian LECLÈRE. *La structure des phrases simples en français : les constructions transitives locatives*. Droz, Genève, 1992. 8.1

[29] IGM. Lesser General Public License for Linguistic Resources. http://igm.univ-mlv/~unitex/lgpllr.html. 1.1

[30] Gaby KLARSFLED and Mary HAMMANI-MC CARTHY. Dictionnaire électronique du ladl pour les mots simples de l'anglais (DELASa). Technical report, LADL, Université Paris 7, 1991. 3.7

[31] Tita KYRIACOPOULOU. *Les dictionnaires électroniques: la flexion verbale en grec moderne*, 1990. Thèse de doctorat. Université Paris 8. 3.7

[32] Tita KYRIACOPOULOU. Un système d'analyse de textes en grec moderne: représentation des noms composés. In *Actes du 5ème Colloque International de Linguistique Grecque, 13-15 septembre 2001*. Sorbonne, Paris, 2002. 3.7

[33] Tita KYRIACOPOULOU, Safia MRABTI, and AnastasiaYANNACOPOULOU. Le dictionnaire électronique des noms composés en grec moderne. *Lingvisticæ Investigationes*, 25(1):7–28, 2002. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[34] Jacques LABELLE. Le traitement automatique des variantes linguistiques en français: l'exemple des concrets. *Lingvisticæ Investigationes*, 19(1):137–152, 1995. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[35] Eric LAPORTE and Anne MONCEAUX. Elimination of lexical ambiguities by grammars : The ELAG system. *Lingvisticæ Investigationes*, 22:341–367, 1998. Amsterdam-Philadelphia : John Benjamins Publishing Company. 7, 7.3

[36] Ville LAURIKARI. TRE home page. http://laurikari.net/tre/. 1.1, 4.7

[37] Annie MEUNIER. *Nominalisation d'adjectifs par verbes supports*, 1981. Thèse de doctorat. Université Paris 7. 8.1

[38] Sun Microsystems. Java. http://java.sun.com. 1.2

[39] Christian MOLINIER and Françoise LEVRIER. *Grammaire des adverbes: description des formes en -ment*. Droz, Genève, 2000. 8.1

[40] Anne MONCEAUX. Le dictionnaire des mots simples anglais : mots nouveaux et variantes orthographiques. Technical Report 15, IGM, Université de Marne-la-Vallée, 1995. 3.7

[41] OpenOffice.org. http://www.openoffice.org. 2.2, 8.2.2

[42] Dong-Ho PAK. *Lexique-grammaire comparé français-coréen. Syntaxe des constructions complétives*. PhD thesis, UQAM, Montréal, 1996. 8.1

[43] Soun-Nam PARK. *La construction des verbes neutres en coréen*, 1996. Thèse de doctorat. Université Paris 7. 8.1

[44] Roger-Bruno RABENNILAINA. *Le verbe malgache*. AUPELF-UREF et Université Paris 13, Paris, 1991. 8.1

[45] Agata SAVARY. *Recensement et description des mots composés - méthodes et applications*, 2000. Thèse de doctorat. Université de Marne-la-Vallée. 3.7

[46] Max SILBERZTEIN. Les groupes nominaux productifs et les noms composés lexicalisés. *Lingvisticæ Investigationes*, 27(2):405–426, 1999. Amsterdam-Philadelphia : John Benjamins Publishing Company. 3.7

[47] Carlos SUBIRATS-RÜGGEBERG. *Sentential complementation in Spanish. A lexico-grammatical study of three classes of verbs*. John Benjamins, Amsterdam/Philadelphia, 1987. 8.1

[48] Thomas TREIG. Complétives en allemand. classification. Technical Report 7, LADL, 1977. 8.1

[49] Lidia VARGA. Classification syntaxique des verbes de mouvement en hongrois dans l'optique d'un traitement automatique. In F. Kiefer, G. Kiss, and J. Pajzs, editors, *Papers in Computational Lexicography (COMPLEX)*, pages 257–265, Budapest, Research Institute for Linguistics, Hungarian Academy of Sciences, 1996. 8.1

[50] Simoneta VIETRI. On the study of idioms in italian. In *Sintassi e morfologia della lingua italiana, Congresso internazionale della Società di Linguistica Italiana*. Roma:Bulzoni, 1984. 3.7

Index